MASTER IN HIGH PERFORMANCE COMPUTING

Development of a computational toolbox to analyse first-passage times and diffusion coefficients in heterogeneous soft-matter system

Supervisor(s):
Prof. Édgar ROLDÁN,
Prof. Ali HASSANALI,
Prof. Ivan GIROTTO

Candidate:
Debarshi BANERJEE

8th EDITION
2021–2022

Contents

List of Figures

List of Algorithms

Acknowledgements

Firstly, I would like to thank my supervisors, Edgar and Ali, for their invaluable help, guidance, and patience throughout this process.
I am indebted to Ivan both for his academic guidance, as well as for his unfaltering support and boundless encouragement, at every step of the way.
I am also extremely thankful to Colin for the many hours he has contributed in debugging the code and explaining basic concepts of MD simulations to me.
I am grateful to my colleagues for discussions and ideas, and to my friends for their company on many a cold, weary evening.
Finally, I would be remiss if I didn't express my immense gratitude for the unconditional love and support I have received from my girlfriend, mother, grandmother, and late maternal grandfather.
Thank you all.

Debarshi Banerjee
September, 2022

Chapter 1

Introduction

1.1 Abstract

The analysis of first-passage time statistics in soft-matter systems, such as water near amino-acid crystals explored in [1], can be vital in understanding the dynamical complexity of their chemical and geometrical properties. From the first-passage time statistics of water molecules, it was shown in [1, 2] that it is possible to infer space-dependent diffusion coefficients in directions normal to various phase boundaries. The analysis developed in [1, 2] is highly-nontrivial, computationally expensive, and system-dependent. Here, in an interdisciplinary collaboration between statistical physics and atomistic simulations, we aim to develop a generic computational methodology which will allow us to extract and analyse trajectories, obtained from molecular dynamics simulations by programs such as GROMACS or LAMMPS, to determine first passage times and spatially resolved diffusion coefficients. We perform exhaustive high-performance-computing benchmarks of our algorithm in various aqueous systems, and develop a user-friendly interface that we will make available to researchers as an open-source toolbox, working on in-silico studies of natural products.

1.2 Background Motivation

Diffusion, in the molecular context, is their net movement, generally from a region of higher concentration to a region of lower concentration. The modern atomistic theory of diffusion was developed by Einstein and Smoluchowski, and later confirmed by the experiments of Perrin. Einstein did a statistical analysis of molecular motion and its effect on particles suspended in a liquid. From this analysis he calculated the mean square displacement of these particles in Ref.[3]. Perrin did a series of experiments, one of which depended on Einstein's calculation of the mean square displacement of suspended particles. His results confirmed Einstein's relation and thus the molecular-kinetic theory.
Biomolecules are mostly present in aqueous environments and therefore their interaction with water governs the behaviour of such systems. In heterogeneous soft-matter systems, (such as water-glutamine, as shown in Fig.2.1), the diffusive dynamics of water is altered significantly in contrast to bulk behaviour. The extent of these dynamical perturbations is thought to play an important role in biological processes and

is therefore of great interest to us.
In Ref.[1], aggregates of glutamine - a protein that plays a crucial role in numerous neurodegenerative diseases is studied. Since protein aggregation usually occurs in aqueous solutions, understanding the solvent's role in the nucleation processes is of vital importance. Recent experiments point to the existence of water pools, whose properties depend on their proximity to the protein fibrils. A quantitative description of the mobility of water close to protein surfaces is therefore invaluable in understanding their behaviour.
Similarly, as we explore later in this thesis, curcumin is a biologically relevant molecule that is widely used for everything from medicinal purposes to food colouring in various parts of the world. It shows very interesting behaviour in an aqueuous environment, namely that the curcumin crystal surfaces are not fully stable. Estimating the change in first-passage times and diffusion coefficients allows us to develop a better understanding of the dynamical evolution of behaviour of such systems. This is also very useful as a diagnostic for equilibration.
From the molecular dynamics simulations of relevant heterogeneous soft-matter interfaces, one can estimate the first-passage times. We perform these simulations using GROMACS to extract first-passage (FP) time statistics of water molecules in different systems. We use the method in Ref.[2] to infer the space-dependent transverse (i.e. in the direction normal to the surface) diffusion coefficient of water as a function of the distance to the interface, purely from the FP time statistics of water. Simpler methodologies like that of using Einstein's mean-square displacement method do not work for heterogeneous systems, and hence, we use the much more computationally demanding method of extracting the diffusion coefficient from the first-passage time analysis of molecular dynamics trajectories.
The theory behind estimating the space-dependent diffusion coefficient purely from first-passage time statistics is explained in Chapter 2, based on Refs.[1, 2]. It is originally developed for Brownian dynamics. In Ref.[2] it was first verified for the Büttiker-Landauer ratchet, and then preliminary applications were made in the domain of molecular systems. In this thesis, we explore further applications of this theory on a wide variety of molecular dynamics simulations.
A major goal of the work that went into this thesis was to allow this computationally expensive analysis to be done in a simple, user-friendly, and efficient way. The trajectory files that are generated by molecular dynamics simulations are many tens of gigabytes in size since one needs a high degree of temporal accuracy to estimate first-passage time statistics. To process them quickly is non-trivial, and it is very important to take advantage of modern parallel architectures to speed up this process wherever feasible.
In this thesis we show the algorithms that were developed, the optimisation techniques and parallelism strategies that were used, and then we present results on a variety of systems already explored in Refs.[1, 2]. We also present preliminary, novel results from the curcumin system mentioned briefly, and then present extensive benchmarking studies of our software on a high-performance computing cluster.

Chapter 2

Theoretical Background

In this chapter we will explore the relationship between the space-dependent diffusion coefficient and first-passage times.

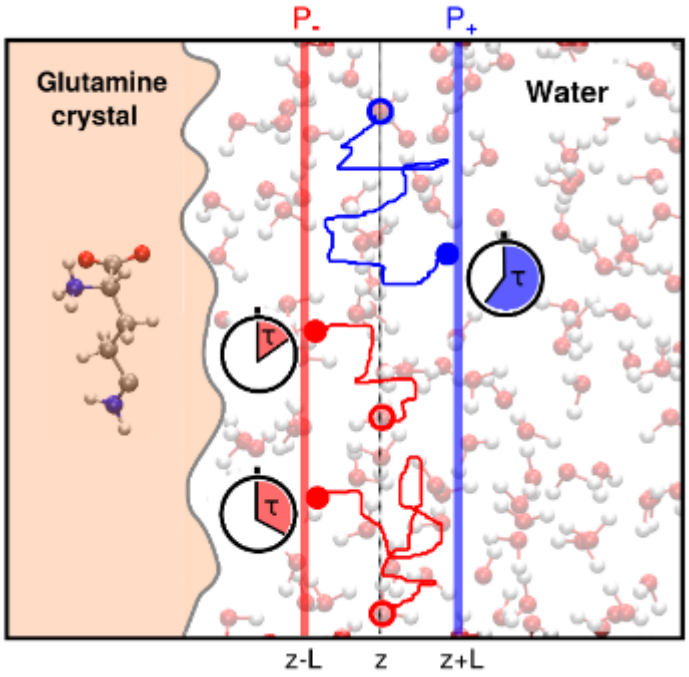
2.1 Concept of first-passage times

A first-passage time (Ref.[4], Ref.[5]) is defined as the time elapsed until a stochastic process first reaches a target state, e.g. the first time when a Brownian particle reaches a spatial region.
Some examples are:

- the first-passage time for one dimensional (1D) Brownian motion to first cross a threshold located at $L > 0$
- the first-passage time for 1D Brownian motion to first escape through any of the two ends of the interval $[L, L]$
- the first-passage time for three-dimensional (3D) Brownian motion to escape a cubic cage $[L, L][L, L][L, L]$

Let us consider an example of a water-glutamine system (Ref.[1]), to illustrate the idea of what a first-passage event exactly is, in systems of our interest. The schematic diagram is shown in Fig.2.1.
Here we consider the 1D first-passage time $\tau(z)$ for a water molecule initially located within a thin shell $[z\delta z/2, z + \delta z/2]$ of width $\delta z > 0$ to cross any of two thresholds located at positions zL and $z + L$, with $L > \delta z$ defined as a half width of the first-passage corridor. The statistics of $\tau(z)$ characterises the kinetics of the ensemble of water molecules at position z near a surface of the glutamine crystal, which can be used to probe the glutamine-water interaction dynamics.
In particular, we focus on the spatial dependency of the mean time $\langle\tau\rangle$ and the passage probabilities to first cross the positive P_+ or the negative P_- threshold. This gives us valuable insight into the change in the diffusive dynamics of water molecules near the interface, and lets us better understand the behaviour of such systems. For further details, see Refs.[1] and [2].
Our approach is to perform molecular dynamics simulations and from the corresponding trajectories, we evaluate three 1D first-passage times $\tau_x(x)$, $\tau_y(y)$, $\tau_z(z)$

Figure 2.1: Visualisation of first-passage events in a water-glutamine crystal interface (Ref.[1]). Near a glutamine crystal (orange shaded area), oxygen atoms of water molecules (open circles) are grouped by their initial positions $z \pm \delta z/2$ into slices of width $\delta z > 0$ along the Z axis, which is perpendicular to the interface plane. Molecules in each group diffuse until they reach boundaries of the corridor $[z - L, z + L]$ at different locations (filled circles) and at a different first-passage time τ (clocks). We measure the times $\tau(z)$ and the probabilities $P_+(z)$ (blue) and $P_-(z)$ (red) that the oxygen atoms reach the absorbing boundary at $z + L$ (blue) or at $z - L$ (red), respectively.



when the oxygen atom of each water molecule travels a distance L from its initial position $q = x, y, z$ along each Cartesian axis X, Y, Z, respectively.
Mathematically, the three first-passage times are defined as follows:

- $\tau_x(x) = \inf\{t > 0; |X(t) - x| \geq L\}$
- $\tau_y(y) = \inf\{t > 0; |Y(t) - y| \geq L\}$
- $\tau_z(z) = \inf\{t > 0; |Z(t) - z| \geq L\}$

with $x = X(0), y = Y(0), z = Z(0)$ the initial values of the X, Y, Z coordinates of the trajectories, respectively.

2.2 Einstein's Formula

The method of mean-squared displacements, which is commonly used to estimate the diffusion constant in homogeneous systems, relies on the Einstein formula for the mean-squared displacement in absence of the potential $U(z)$

$$\langle [z(t) - z(0)]^2 \rangle = 2D_0 t \tag{2.1}$$

The mean first-passage time depends on L quadratically, i.e., $\langle \tau \rangle \propto L^2$. The same scaling law ($\sim L^2$) is found for the mean first-passage time of 1D Brownian motion.
In a 1D Brownian motion the probability density $P(z,t)P(z,t|z_0,0)$ for a coordinate z of a molecule moving with diffusion coefficient D_0 evolves according to a Fokker-Planck equation:

$$\partial_t P(z,t) = D_0 \partial_z^2 P(z,t) \tag{2.2}$$

In this case a probability distribution of the mean first-passage time (i.e. time elapsed) for a molecule to escape the interval $[z_0 L, z_0 + L]$ is known and its mean value reads:

$$\langle \tau_{bulk} \rangle = \frac{L^2}{2D_0} \tag{2.3}$$

From Eq.(2.3) and Eq.(2.1), we can see that D_0 may be extracted using 3 different statistics - the Mean First-Passage Times (MFPT) and Mean Square Displacements (MSD).
This method is suitable for a homogeneous 'bulk water' system, and in fact, we reproduce this quadratic dependence between the mean-first passage time and the first passage window length (L) in a future section.
It is not suitable for measuring the transverse diffusion coefficient in heterogeneous systems which possess some net potential.

2.3 Inferring diffusion coefficient from first-passage statistics

We consider the first-passage time of a molecule, which escapes a symmetric interval of length $2L$ centred at the initial position $z_0 = z(0)$. This first-passage time is formally defined as

$$\tau(z_0) = \inf \{ t > 0 : |z(t) - z_0| \geq L \} \tag{2.4}$$

For each trajectory, $z(t)$, $\tau(z_0)$ can take a different positive, random value. Therefore, the mean first-passage time is

$$\langle \tau(z_0) \rangle = \int_0^\infty \tau P[\tau(z_0) = \tau] d\tau \tag{2.5}$$

where $P[\tau(z_0) = \tau]$ is the probability that the particle exits from the spatial region of interest, $[z_0 - L, z_0 + L]$, within a time interval $[\tau, \tau + d\tau]$.
The first-passage events defined above can be characterised as positive and negative (see Fig 1.1), depending on whether the diffusing molecule escapes through the positive or negative end of the interval $[z_0 L, z_0 + L]$, respectively. In our system, particles always exit from the region of interest in a finite time, i.e., $\tau(z_0) < \infty$ for all trajectories, therefore, we can also consider

$$P_+(z_0) + P_-(z_0) = 1 \tag{2.6}$$

Since we are interested in measuring the space-dependent diffusion coefficient in the presence of a net potential, where the method of mean square-displacements doesn't

work well, here we consider a model of a one-dimensional inhomogeneous diffusion along the Z axis. The space-dependent diffusion coefficient is $D(z)$ and this process occurs in the presence of a net potential $U(z)$.
In this model, the evolution of the density of the fluid $P(z,t)$, which depends on the spatial coordinate z and time t, obeys the Smoluchowski equation:

$$\frac{\partial P(z,t)}{\partial t} = \frac{\partial}{\partial z}\left[\beta D(z)P(z,t)\frac{\partial U(z)}{\partial z} + D(z)\frac{\partial P(z,t)}{\partial z}\right] \tag{2.7}$$

where $\beta = (k_B T)^{-1}$, with k_B as Boltzmann's constant and T the temperature of the fluid. Here we have also assumed Einstein's relation for the mobility, $\mu(z) = \beta D(z)$.
We consider here the escape of a molecule with the initial position $z(0) = z$ through one of the two boundaries of the passage corridor $[zL, z+L]$ with $L > 0$. The probability $P_{(z)}$ and $P_+(z)$ for the system to first reach the boundary at zL and $z+L$, respectively, are given by

$$P_-(z) = \frac{\int_z^{z+L} dy\, e^{-\beta U(y)}/D(y)}{\int_{z-L}^{z+L} dy\, e^{-\beta U(y)}/D(y)} \tag{2.8}$$

$$P_+(z) = \frac{\int_{z-L}^{z} dy\, e^{-\beta U(y)}/D(y)}{\int_{z-L}^{z+L} dy\, e^{-\beta U(y)}/D(y)} \tag{2.9}$$

The mean first-passage time can then be written as

$$\langle \tau(z) \rangle = \int_z^{z+L} \frac{dy}{D(y)} \int_{z-L}^{y} dx\, e^{-\beta[U(x)-U(y)]} - P_-(z) \int_{z-L}^{z+L} \frac{dy}{D(y)} \int_{z-L}^{y} dx\, e^{-\beta[U(x)-U(y)]} \tag{2.10}$$

Equations (2.8), (2.9), and (2.10) can only be solved analytically for some special cases. For example, for a sufficiently small L, we may approximate the effective potential by a truncated power-series expansion

$$U(y) \simeq U(z) + (y-z)h + O(y^2) \tag{2.11}$$

which holds for $-L < y - z < L$, where we take $h = U'(z)$.
Because the diffusion coefficient must be strictly positive, we can assume the following approximation

$$D(y) \simeq D(z)e^{\kappa(y-z)+O(y^2)} \tag{2.12}$$

where $\kappa = \frac{D'(z)}{D(z)}$
Therefore, Eq.(2.12) can also be written as

$$\ln D(y) \simeq \ln D(z) + \kappa(y-z) + O(y^2) \tag{2.13}$$

Substituting Eq.(2.12) and Eq.(2.13) into Eq.(2.8) and Eq.(2.9), we can obtain the expression for the passage probabilities as

$$P_-(z) \simeq \frac{1}{1+e^{-L(\beta h-\kappa)}} \tag{2.14}$$

$$P_+(z) \simeq \frac{1}{1+e^{L(\beta h-\kappa)}} \tag{2.15}$$

Substituting Equations (2.11), (2.12), (2.14), and (2.15) into Eq.(2.10), we obtain

$$\langle \tau(z) \rangle \simeq \frac{\cosh[(\beta h + \kappa)L/2] / \cosh[(\beta h - \kappa)L/2] - 1}{\kappa \beta h D(z)} \tag{2.16}$$

Now, let's consider the limit of $\kappa \to 0$. This is actually the same as $D(y) \simeq D(z)$. Physically this can be interpreted as the regime where the diffusion coefficient has little to no variability with position. Using this limit, and Eq.(2.11), we obtain

$$\langle \tau(z) \rangle \simeq \frac{L \tanh(L\beta h/2)}{D(z)\beta h} \tag{2.17}$$

In addition, in the limit of $h \to 0$, i.e., where the effective potential is basically constant, we obtain

$$\langle \tau(z) \rangle \simeq \frac{L^2}{2D(z)} \tag{2.18}$$

This is the same as Eq.(2.3) and this gives us the formula for the mean first-passage time in bulk water. We therefore recover the quadratic dependency of the mean first-passage time on L, obtained in the bulk.
As per Ref.[2], we can write the relationship between $U(z)$ and $\tilde{U}(z)$, the Smoluchowski and Fokker-Planck potentials respectively, as

$$\tilde{U}(z) = U(z) - k_B T \log[D(z)/D_0] \tag{2.19}$$

The passage probabilities of the positive and negative first-passage events are thus obtained by substituting Eq.(2.19) in Eq.(2.8) and Eq.(2.9).

$$P_-(z) = \frac{\int_z^{z+L} dy\, e^{-\beta \tilde{U}(y)}}{\int_{z-L}^{z+L} dy\, e^{-\beta \tilde{U}(y)}} \tag{2.20}$$

$$P_+(z) = \frac{\int_{z-L}^{z} dy\, e^{-\beta \tilde{U}(y)}}{\int_{z-L}^{z+L} dy\, e^{-\beta \tilde{U}(y)}} \tag{2.21}$$

Eq.(2.20) and Eq.(2.21), which were obtained for the Fokker-Planck model, are similar to Eq.(2.8) and Eq.(2.9), which were obtained for the Smoluchowski model. The only difference is that we consider the Fokker-Planck potential ($\tilde{U}$), instead of the Smoluchowski potential (U).
Just like in the previous treatment, we assume that over the small interval $[z - L, z + L]$, the diffusion coefficient, and the slope of the Fokker-Planck potential are approximately constant. Therefore, for $y \in [z - L, z + L]$, we assume that

$$D(y) \simeq D(z) + O(y - z) \tag{2.22}$$

$$\partial_z \tilde{U}(y) \simeq \partial_z \tilde{U}(z) + O(y - z) \tag{2.23}$$

To find the diffusion coefficient, we use the formula for the mean exit time from Eq.(2.10) and proceed as we did previously.

$$\langle \tau(z) \rangle \simeq \frac{k_B T L}{D(z)\partial_z \tilde{U}(z)} \tanh\left[\frac{L \partial_z \tilde{U}(z)}{2 k_B T}\right] \tag{2.24}$$

Dividing Eq.(2.21) by Eq.(2.20) and using Eq.(2.23), we obtain

$$\partial_z \tilde{U}(z) \simeq \frac{k_B T}{L} \ln \left[\frac{P_-(z)}{P_+(z)} \right] \tag{2.25}$$

From Eq.(2.24) and Eq.(2.25), we finally obtain the expression for the diffusion coefficient as

$$D(z) \simeq \frac{L^2}{\langle \tau(z) \rangle} \frac{P_+(z) - P_-(z)}{\ln[P_+(z)/P_-(z)]} \tag{2.26}$$

The beauty of Eq.(2.26) is that it means that we can calculate the diffusion coefficient purely in terms of first-passage time statistics (specifically, the positive and negative passage probabilities) extracted from molecular dynamics simulations or even experimental data collection.

Chapter 3

Details of PyFPT

3.1 Overall Structure

In the remainder of this thesis, we show the validity of the theory explored in Chapter 2 for different systems, using a simple to use, efficient, and accurate toolbox that we developed. The code is in Python and can be run in a user-friendly manner by someone with almost no programming experience.
In the interest of this goal, we have a plain-text `params.py` file that can be trivially edited by someone with the following information

- path to input trajectory file
- path to input topology file
- `dx` - width of bins
- `L` - first-passage window width
- minimum sampling criteria
- time interval to wait for 2 uncorrelated events
- the axes to process (axis of binning, axis of MFPT estimation)
- the spatial range of interest (in Angstrom)
- the range of frames of the simulation (corresponds to time)
- details about the interface (if any)
- further parameters for plotting functions

Finally, once this is done the code can be run with a simple `python3 main.py 4`, where 4 can be replaced by the number of CPUs you want to use. If you choose an unreasonable number, there are some automated checks to limit that as well.
The directory structure of the project is:

```
dbanerjee@mintbox ~/github/PyFPT (main u=) tree -d -I '*water*|curcumin'

 benchmarks
 docs
 examples
 fpt
     diffusion
     gdi
     mfpt
     plots
     tests
     utilities
     params.py
     main.py
```

Since the code is run with Python, it is best to have a sanitized conda environment for running the code. The following script (also present in the repository), takes care of that (assuming you have anaconda installed).

```
conda create --name fpt -y
conda init bash
source ~/.bashrc
conda activate fpt
conda config --add channels conda-forge
conda install numba numpy scipy matplotlib notebook mypy dask -y
pip install --upgrade MDAnalysis
pip install --upgrade MDAnalysisTests
```

This creates a conda environment - `fpt`, which can be activated using `conda activate fpt`, and will possess all the necessary packages.
The code has been tested on Python 3.8 and 3.10.
Some examples of `params.py` files for the systems that we explore in this thesis are in the `examples` directory to give a user a good starting point.

3.2 Details of MDAnalysis

A key component that we use in our software is the Python package - `MDAnalysis` [6, 7]. This is a well-developed module that allows someone to read a variety of different molecular dynamics simulation trajectory formats and perform various convenient transformations on them. Not only do we obtain the details of the positions of each atom at every time-step of the simulation, we also obtain other vital details, such as `dt` (time interval of data), the total number of frames, the size of the system (in nanometers or Angstrom) etc. We make extensive use of MDAnalysis to allow our software to work with any trajectory generated by a variety of common MD programs.
The way it works is by importing the trajectory and topology files to create a `Universe` of atoms.

```
import MDAnalysis as mda
uni = mda.Universe(output.xtc, topol.top)
```

Next, we select a subset of atoms we are interested in to create an `AtomGroup`. In our case, since we are mostly interested in Water diffusion, the default in `params.py` is set to `OW`, which refers to the Oxygen atoms in each Water molecule. However, this can be trivially changed by editing the `params.py` file with the residue ID of interest.

```
atom_group = uni.select_atoms("name OW")
```

From here onwards, we have access to all the information we need

```
atom_group.n_atoms                              # number of atoms
atom_group.n_frames                             # number of frames
atom_group.dt                                   # time interval
atom_group.positions                            # positions array
atom_group.universe.trajectory[10]              # set to 10th frame
atom_group.universe.trajectory.ts.dimensions    # dimensions of the box
```

In summary, this section was intended to give some insight into why we use Python, as this package makes it easy to deal with a wide variety of MD generated data formats, whether binary or uncompressed, and therefore, it fulfils the originally stated goal of being widely and easily accessible to the larger computational chemistry/physics community, even amongst those with limited programming experience. Also this section helps people be more familiar with some of the terminology (particularly, `atom_group`), that we will use in various algorithms in the next sections.

3.3 Algorithms to estimate the first-passage time

In this section, I present 2 algorithms, both of which are similar, that we use to estimate the mean first-passage time, the diffusion coefficient, and other associated statistics.
Algorithm 1 is the default algorithm in our code. In summary, it is when we have read the data into memory prior to usage, and therefore, it is much faster. However, it is memory inefficient, and depending on the requirements of the user, especially if they are working on low-resource systems, they may prefer to use Algorithm 2.

Algorithm 1 MFPT Calculation after reading data into memory

Require: *coords* ▷ Position coordinates of all atoms at all times
Require: *axis* ▷ Axis of interest
Require: *atom_start*, *atom_end* ▷ Range of atoms to process
Require: *atom_step* ▷ How many atoms to skip over - useful for parallelising
Require: *z_start*, *z_end* ▷ Spatial range of processing
Require: *dz*, *L* ▷ Parameters needed for FP time estimation
Require: *frame_start*, *frame_end* ▷ Temporal range of processing
Require: *frame_step* ▷ Skip some frames to check when FP event occurs
Require: *corr_time* ▷ Wait for statistically uncorrelated events
Require: *dims* ▷ Array with box coordinates
Require: *dt* ▷ Time interval for data in trajectory files
Require: *min_events* ▷ minimum number of FP events needed for sampling
Ensure: $L > dz$

$bins \leftarrow linspace(z\_start, z\_end, step = dz)$
for each chosen atom (j) **do**
    $curr\_pos \leftarrow coords[0, j, axis]$
    $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos//dim)$ ▷ // represents floor division
    if *curr_pos_pbc* outside range of z_start, z_end **then**
        $check\_exit\_range \leftarrow True$
    end if
    $starting\_pos \leftarrow curr\_pos$
    $starting\_pos\_pbc \leftarrow curr\_pos\_pbc$
    $starting\_time \leftarrow dt * frame\_start$
    while we iterate over the trajectory in steps of $frame\_step$ (i) **do**
        $curr\_time \leftarrow dt * ((i * frame\_step) + frame\_start)$
        $curr\_pos \leftarrow coords[i, j, axis]$
        $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos//dim)$
        Skip the loop's current iteration if it outside the spatial range z_start, z_end
        Wait for *corr_time* interval after it comes back in range OR after a FP event is completed
        $dist = curr\_pos - starting\_pos$
        $dist\_abs = abs(dist)$
        if $dist\_abs > L$ **then**
            $bin\_index \leftarrow int((starting\_pos - z\_start)//dz)$
            $total\_events[bin\_index] \mathrel{+}= 1$
            if $dist > 0$ **then**
                $FP_{+}[bin\_index] \mathrel{+}= 1$
            else
                $FP_{-}[bin\_index] \mathrel{+}= 1$
            end if
            $final\_time \leftarrow curr\_time$
            $FP\_time \leftarrow final\_time - starting\_time$
            $MFPT[bin\_index] \mathrel{+}= FP\_time$
            $starting\_time \leftarrow final\_time$ ▷ Reset starting times and positions
            $starting\_pos \leftarrow current\_pos$
            $starting\_pos\_pbc \leftarrow current\_pos\_pbc$
        end if
    end while
end for
cond is the condition for bins with atleast *min_events* in both + and - directions
delete all other bins in all the data arrays
$MFPT \leftarrow MFPT/total\_events$ ▷ vectorized division: calculates mean FP time for each corresponding bin by dividing by total number of events in that bin
cond_fpt is the condition for $MFPT! = 0$, $P_{+} \neq P_{-}$ and $(P_{+}, P_{-})! = 0$
$D\_fpt[NOT\ cond\_fpt] \leftarrow L^2/(2\tau)$
$D\_fpt[cond\_fpt] \leftarrow [L^2/(\tau)]\,(P_{+} - P_{-})/\ln(P_{+}/P_{-})$

In Algorithm 2, we use MDAnalysis to read the data from the disk on the fly, where it stores the offsets. This is much slower, but on the other hand, it is memory efficient.

Algorithm 2 MFPT Calculation directly from trajectory files

Require: $atom\_group$ ▷ Atom Group of interested atom
Require: $axis$ ▷ Axis of interest
Require: $atom\_start, atom\_end$ ▷ Range of atoms to process
Require: $atom\_step$ ▷ How many atoms to skip over - useful for parallelising
Require: z_start, z_end ▷ Spatial range of processing
Require: dz, L ▷ Parameters needed for FP time estimation
Require: $frame\_start, frame\_end$ ▷ Temporal range of processing
Require: $frame\_step$ ▷ Skip some frames to check when FP event occurs
Require: $corr\_time$ ▷ Wait for statistically uncorrelated events
Require: min_events ▷ minimum number of FP events needed for sampling
Ensure: $L > dz$

$dims \leftarrow atom\_group.universe.trajectory.ts.dimensions$
$dt \leftarrow atom\_group.universe.trajectory.dt$
$bins \leftarrow linspace(z\_start, z\_end, step = dz)$
for each chosen atom (j) **do**
    $atom\_group \leftarrow atom\_group.universe.trajectory[frame\_start]$ ▷ The trajectory should start from the desired frame
    $curr\_pos \leftarrow atom\_group.positions[0, j, axis]$
    $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos//dim)$ ▷ // represents floor division
    if $curr\_pos\_pbc$ outside range of z_start, z_end **then**
        $check\_exit\_range \leftarrow True$
    end if
    $starting\_pos \leftarrow curr\_pos$
    $starting\_pos\_pbc \leftarrow curr\_pos\_pbc$
    $starting\_time \leftarrow dt * frame\_start$
    while we iterate over the trajectory in steps of $frame\_step$ (i) **do**
        $curr\_time \leftarrow atom\_group.universe.trajectory.time$
        $curr\_pos \leftarrow atom\_group.positions[i, j, axis]$
        $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos//dim)$
        Skip the loop's current iteration if it outside the spatial range z_start, z_end
        Wait for $corr\_time$ interval after it comes back in range OR after a FP event is completed
        $dist = curr\_pos - starting\_pos$
        $dist\_abs = abs(dist)$
        if $dist\_abs > L$ **then**
            $bin\_index \leftarrow int((starting\_pos - z\_start)//dz)$
            $total\_events[bin\_index] \mathrel{+}= 1$
            if $dist > 0$ **then**
                $FP_{+}[bin\_index] \mathrel{+}= 1$
            else
                $FP_{-}[bin\_index] \mathrel{+}= 1$
            end if
            $final\_time \leftarrow curr\_time$
            $FP\_time \leftarrow final\_time - starting\_time$
            $MFPT[bin\_index] \mathrel{+}= FP\_time$
            $starting\_time \leftarrow final\_time$ ▷ Reset starting times and positions
            $starting\_pos \leftarrow current\_pos$

35: $starting\_pos\_pbc \leftarrow current\_pos\_pbc$
36: **end if**
37: **end while**
38: **end for**
39: *cond* is the condition for bins with atleast *min_events* in both + and - directions
40: delete all other bins in all the data arrays
41: $MFPT \leftarrow MFPT/total\_events$ ▷ vectorized division: calculates mean FP time for each corresponding bin by dividing by total number of events in that bin
42: $cond\_fpt$ is the condition for $MFPT! = 0$, $P_+ \neq P_-$ and $(P_+, P_-)! = 0$
43: $D\_fpt[NOT\ cond\_fpt] \leftarrow L^2/(2\tau)$
44: $D\_fpt[cond\_fpt] \leftarrow [L^2/(\tau)]\,(P_+ - P_-)/\ln(P_+/P_-)$

In terms of logic this is broadly the same as Algorithm 1, but I wrote it separately since it uses MDAnalysis to read from the disk, so the way the information is obtained and used is different. Even though the differences are minute, they are important to note, particularly because of the large associated performance penalty.

3.4 Reading data into memory to improve performance

Like we describe in Algorithm 1, we need to read the entire simulation data into the memory prior to usage. Or more practically, depending on our parallelisation strategy, we need to read in the data of the atoms that a particular process will be in-charge of.

Algorithm 3 Reading MD simulation data into memory

Require: *atom_group* ▷ The group of atoms whose diffusivity we are interested in
Require: $frame\_start$
Require: $frame\_end$
Require: $frame\_step$ ▷ Skip some frames to check when a FP event occurs
Require: *atom_start* ▷ Starting atom for reading in data - Useful in parallelisation
Require: *atom_end*
Require: *atom_step* ▷ Linked to number of processors demanded for parallelisation
box_dims ← $atom\_group.universe.trajectory.ts.dimensions$
dt ← $atom\_group.universe.trajectory.dt$
atom_pos ← $atom\_group.positions$
n_frames ← $int((frame\_end - frame\_start)/frame\_step)$
n_atoms ← $atom\_pos.shape[0]$
Ensure: pos_data.shape = (n_frames, n_atoms/atom_step, 3)
for each atom in range frame_start, frame_end, intervals of frame_step (i) **do**
    pos_data[i, ...] = atom_group.positions[atom_start:atom_end:atom_step, :]
end for
return dt, box_dims, pos_data

3.5 Algorithm to calculate the Centre-of-Mass at every time step

If the centre of mass of the molecules is simply determined as the centre of mass within the unit cell, the jump of a molecule over the periodic boundary will result in discontinuities in the constraint force.
In simpler terms, for systems with periodic boundary conditions (PBCs), and unstable interfaces, the COM moves around and we need to define a new, simple method to stabilise it at every time-step.
The following is a simple algorithm based on Ref.[8] that is necessary to be run at every time-step prior to estimating the FPT statistics for systems where the interface shifts a lot.
It is crucial for performance reasons that this algorithm works efficiently, since at every time-step, prior to doing the FPT processing, it will be called and we will be returning the new coordinates to the main `fpt()` function.
Hence, this function - `pbc_com()`, is optimized by using Numba. Since the operations here are purely numeric calculations on large Numpy arrays, Numba is perfectly suited for the task, where we use the `@njit` decorator to the function, so that it is pre-compiled.

Algorithm 4 COM repositioning for PBC

Require: $coords$ ▷ The position coordinates of all atoms
Require: box_dims ▷ Dimensions of the box
Require: $mass$ ▷ Since we measure diffusivity of only one molecule, its mass is enough

$angles \leftarrow (2\pi) coords / box\_dims$
$sinsum \leftarrow sum(mass * \sin(angles))$
$cossum \leftarrow sum(mass * \cos(angles))$
$new\_com \leftarrow \arctan(sinsum/cossum) * mass/(2\pi)$
$tmp\_coords = (coords \% box\_dims) - new\_com$
$new\_coords = ((tmp\_coords + (box\_dims/2)) \% box\_dims) - (box\_dims/2)$
return new_coords

3.6 Algorithm to calculate diffusion coefficient analytically

The following algorithm is for estimating the diffusion coefficient iteratively parsing the trajectory. In simple terms, it measures the mean distance travelled by the molecule of interest at pre-specified time intervals, within the spatial and temporal range of interest. It is particularly useful for comparing our FPT estimated diffusion coefficient to, especially in the bulk region.

Algorithm 5 Diffusion coefficient estimation analytically

Require: $coords$ ▷ Position coordinates of all atoms at all times
Require: $axis$ ▷ Axis of interest
Require: $atom\_start, atom\_end$ ▷ Range of atoms to process
Require: $atom\_step$ ▷ How many atoms to skip over - useful for parallelising

Require: z_start, z_end ▷ Spatial range of processing
Require: dz, L ▷ Parameters needed for FP time estimation
Require: $frame\_start, frame\_end$ ▷ Temporal range of processing
Require: $frame\_step$ ▷ Skip some frames to check when FP event occurs
Require: $dims$ ▷ Array with box coordinates
Require: dt ▷ Time interval for data in trajectory files
Require: min_events ▷ minimum number of FP events needed for sampling
Ensure: $L > dz$

$bins \leftarrow linspace(z\_start, z\_end, step = dz)$
for each chosen atom (j) **do**
    $curr\_pos \leftarrow coords[0, j, axis]$
    $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos // dim)$ ▷ // represents floor division
    if $curr\_pos\_pbc$ outside range of z_start, z_end **then**
        $check\_exit\_range \leftarrow True$
    end if
    $starting\_pos \leftarrow curr\_pos$
    $starting\_pos\_pbc \leftarrow curr\_pos\_pbc$
    $starting\_time \leftarrow dt * frame\_start$
    while we iterate over the trajectory in steps of $frame\_step$ (i) **do**
        $curr\_time \leftarrow dt * ((i * frame\_step) + frame\_start)$
        $curr\_pos \leftarrow coords[i, j, axis]$
        $curr\_pos\_pbc \leftarrow curr\_pos - dim * (curr\_pos // dim)$
        Skip the loop's current iteration if it outside the spatial range z_start, z_end
        $dist = curr\_pos - starting\_pos$
        $dist\_abs = abs(dist)$
        $final\_time \leftarrow curr\_time$
        $bin\_index \leftarrow int((starting\_pos - z\_start) // dz)$
        $total\_events[bin\_index] \mathrel{+}= 1$
        if $final\_time = starting\_time$ **then**
            $D = 0$
        else
            $D = (dist\_abs^2)/(2(final\_time - starting\_time))$
        end if
        $starting\_time \leftarrow final\_time$ ▷ Reset starting times and positions
        $starting\_pos \leftarrow current\_pos$
        $starting\_pos\_pbc \leftarrow current\_pos\_pbc$
    end while
end for
$D \leftarrow D/total\_events$ where $total\_events \neq 0$

Chapter 4

Analysis of Various Systems

4.1 Bulk Water

In this section we reproduce results for a simple bulk water system from Ref.[1]. This is a homogeneous system where we expect to find isotropic behaviour since it is just a box of water molecules. We expect to see the quadratic dependence between the mean first-passage time (τ) and the first passage window length (L) of Eq.(2.3) as well.

4.1.1 MD simulation details

We use the SPC-E (Simple Point Charge Model with Effective Polarization Correction) as per Ref [9]. There are 1019 water molecules in a box of dimensions $[3.1119, 3.1119, 3.1119]$ nm.
We use the following parameters for the simulation.

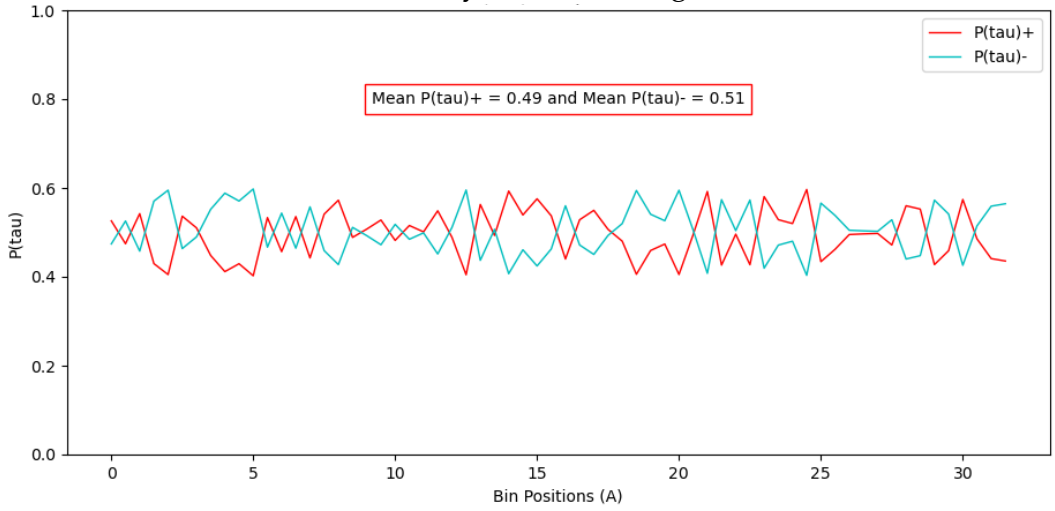
- integrator = md (leap-frog algorithm)
- dt = 1 fs
- nsteps = 2 million
- total duration = 2 ns
- frequency of printing = 10 steps = 10 fs
- T = 298 K
- Thermostat = Velocity-Rescaling [10]
- Time constant for coupling = 0.1 ps
- PBC = X,Y,Z

For the FP analysis, we throw away the first 0.5 ns to ensure a well-equilibriated system.

4.1.2 FP analysis

We present the comparison of $P_+(\tau)$ and $P_-(\tau)$ which are the probabilities of moving either along +ve X-axis or -ve X-axis, respectively. This simulation is done for $dx = 0.1$ Å and $L = 1.0$ Å. In the following section we see the evolution of the absolute values of the mean first-passage time (MFPT) with increase in L for a constant dx.
Here, we mainly see one specific case of the comparison of the probabilities. In the bulk they should be approximately equal in either direction, and if one pays attention to the scale along the Y-axis of the following graph, we clearly see that the variation is minimal (from 0.48 to 0.52) - and the mean values for both $P_+(\tau)$, $P_-(\tau) \approx 0.5$ as one would expect. This is a good consistency check for our theoretical model.

Figure 4.1: The comparison between $P_+(\tau), P_-(\tau)$ for the bulk water system. It is expected to be around 0.5 since the system is homogeneous.



4.1.3 MFPT vs L - Dependence

As mentioned earlier, we expect to see the quadratic dependence between the mean first-passage time (τ) and the first passage window length (L) of Eq.(2.3). Since the system is bulk water and therefore isotropic, we expect to recover this dependence along all independent axes. For all of these cases, we chose $dx = 0.1$ Å. The L was varied suitably as needed, in increments of 1 Å from 1 Å to 1 nm.

Figure 4.2: The quadratic dependence for MFPT vs L for the X axis. We estimate $D_x = 2.90 \pm 0.08\, nm^2/ns$.

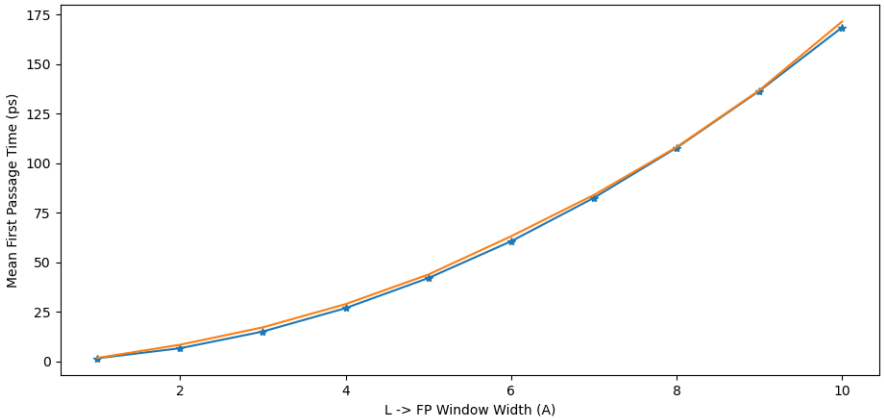


Figure 4.3: The quadratic dependence for MFPT vs L for the Y axis. We estimate $D_y = 2.69 \pm 0.11\, nm^2/ns$.

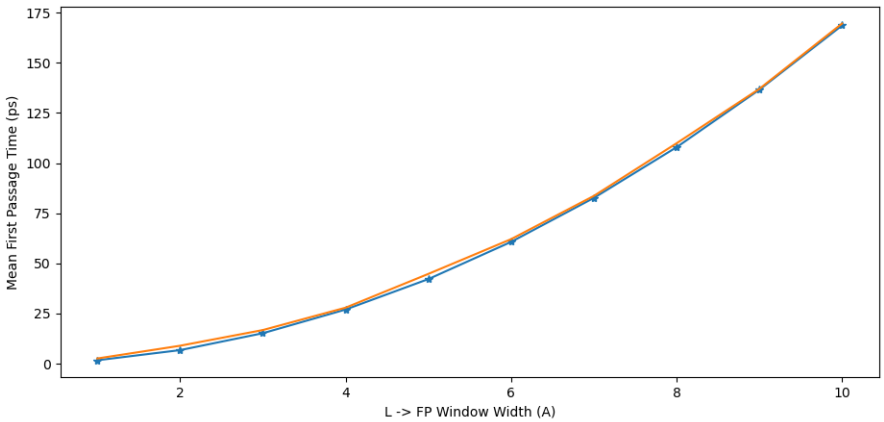
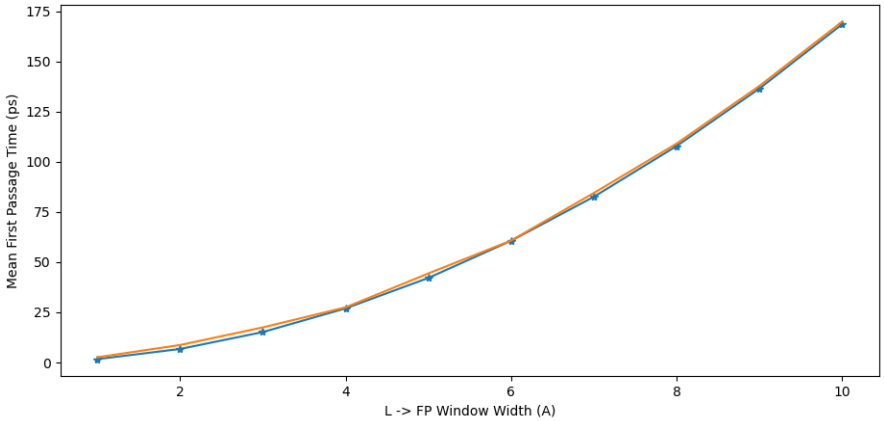


Figure 4.4: The quadratic dependence for MFPT vs L for the Z axis. We estimate $D_z = 2.87 \pm 0.08\, nm^2/ns$.



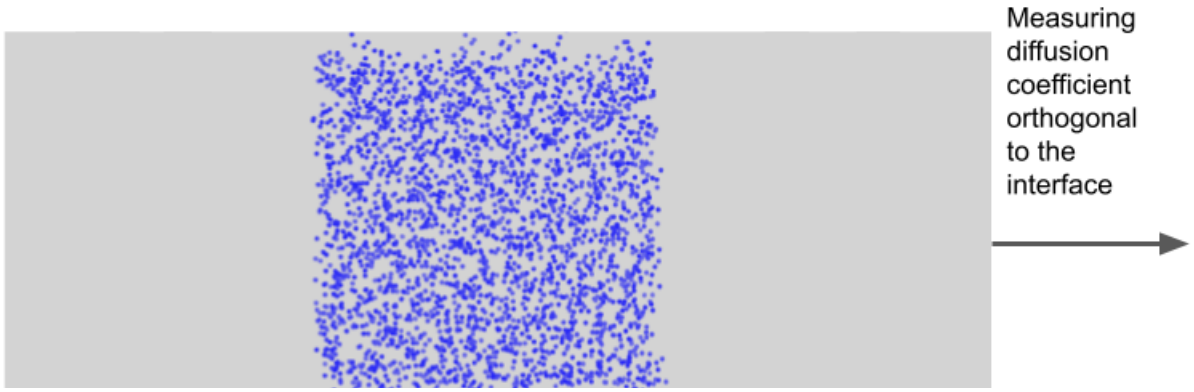
The 3 spatial-dependent diffusion coefficient values that we obtain are in the range $D = 2.60 - 2.90, nm^2/ns$. These are consistent with values of Ref. [11] which pre-

dict values from $2.6 - 5.1\,nm^2/ns$ depending on the water model that is used. The experimental value of the diffusion coefficient of water in bulk is $2.30\,nm^2/ns$.
The diffusion coefficient predicted by our method is in line with established values in literature, and we can be satisfied of the validity of our theoretical predictions for bulk water. The quadratic dependence mentioned earlier is recovered as well, and this is further evidence in support of the theoretical model.

4.2 Water-Air interface

In this section we reproduce results for a simple water-air interface system from Ref.[2]. In practice, it is the same as the bulk water system, with one axis of the box extended arbitrarily from 3.1119 nm to 100.0 nm, and this gap is considered as air since it basically should prevent water diffusion along that axis. To be more accurate, it should be considered the "water-vacuum" interface. The other 2 axes are the same as before, and we have periodic boundary conditions in all 3 directions. This is a model appropriate for a consistency check of our theory for interface-based systems.

Figure 4.5: Visualising the Water-Air Interface. The water molecules are shown in blue, and the grey background represents the "air", which is basically increasing the dimensions of the simulation box in one direction, and leaving it empty.



4.2.1 MD simulation details

We use the SPC-E (Simple Point Charge Model with Effective Polarization Correction) as per Ref [9]. There are 1019 water molecules in a box of dimensions $[3.1119, 3.1119, 100.0]$ nm.
We use the following parameters for the simulation.

- integrator = md (leap-frog algorithm)
- dt = 1 fs
- nsteps = 2 million
- total duration = 2 ns
- frequency of printing = 10 steps = 10 fs

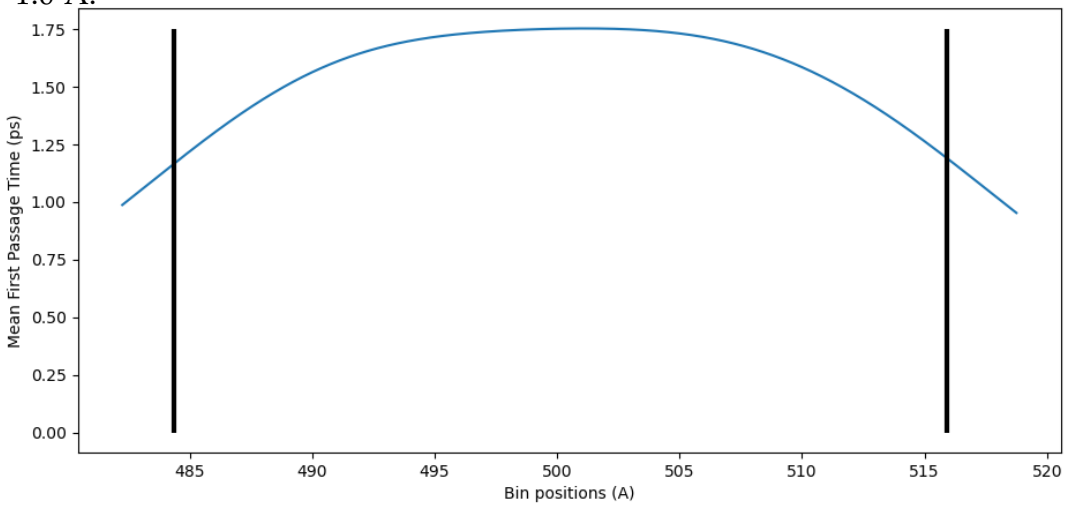
- T = 298 K
- Thermostat = Velocity-Rescaling [10]
- Time constant for coupling = 0.1 ps
- PBC = X,Y,Z

For the FP analysis, we throw away the first 0.5 ns to ensure a well-equilibriated system.

4.2.2 FP analysis

This simulation is done for $dx = 0.5$ Å and $L = 1.0$ Å. In Fig.4.6, we see the mean first-passage time of water and as one would expect from a physical standpoint, the value in the bulk is similar to the value for Bulk Water for $L = 1.0$ Å. As we near the air interface, FP times get lesser since it is less favourable for water to exist near this interface, and so it is forced to "run" or diffuse faster, hence, FP times reduce.

Figure 4.6: Estimating the mean first-passage time for the water-air interface. The black bars represent the estimated Gibbs Dividing Interface (GDI). The blue line is drawn connecting the points that represent the MFPT in each bin. In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å.



In the bulk the probability of FP events along either axis ≈ 0.5 as one would expect from our findings in Section 4.1. As we near the interface, we see that water molecules prefer to diffuse faster away from the interface, and towards the bulk. Hence, the probability of moving right, i.e., along +ve X axis increases a lot as we near the left interface, and vice-versa. This behaviour is clearly seen in Fig.4.7.

Figure 4.7: Comparing $P_+(\tau), P_-(\tau)$ for the water-air interface. The black bars represent the estimated Gibbs Dividing Interface (GDI). The blue and red lines signify the passage probabilities in -Z and +Z direction respectively.

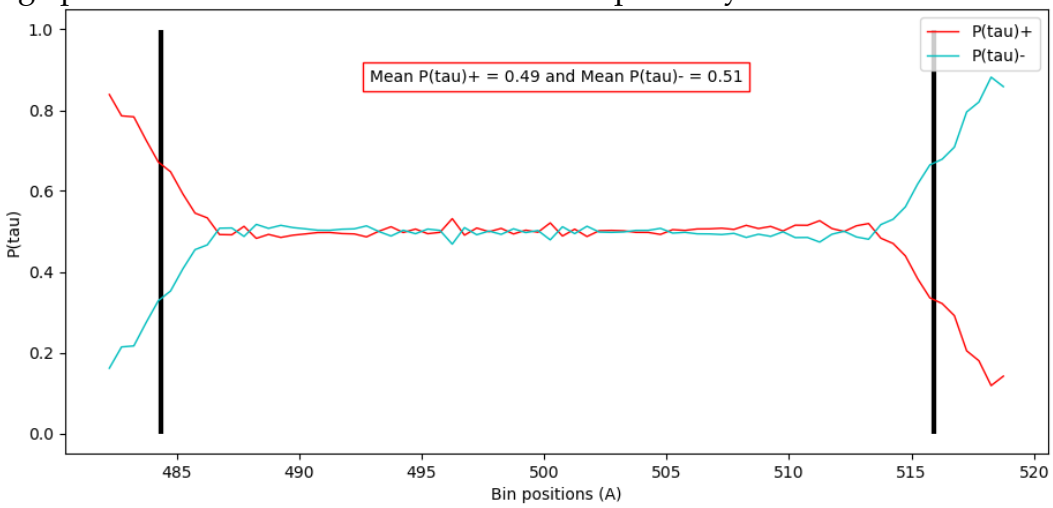


Figure 4.8: Estimating the space-dependent transverse diffusion coefficient analytically using Alg.5 for the water-air interface. The red bars represent the estimated Gibbs Dividing Interface (GDI).The blue line is drawn connecting the points that represent the D_z value in each bin. The estimated mean diffusion coefficient is $3.38 nm^2/ns$.

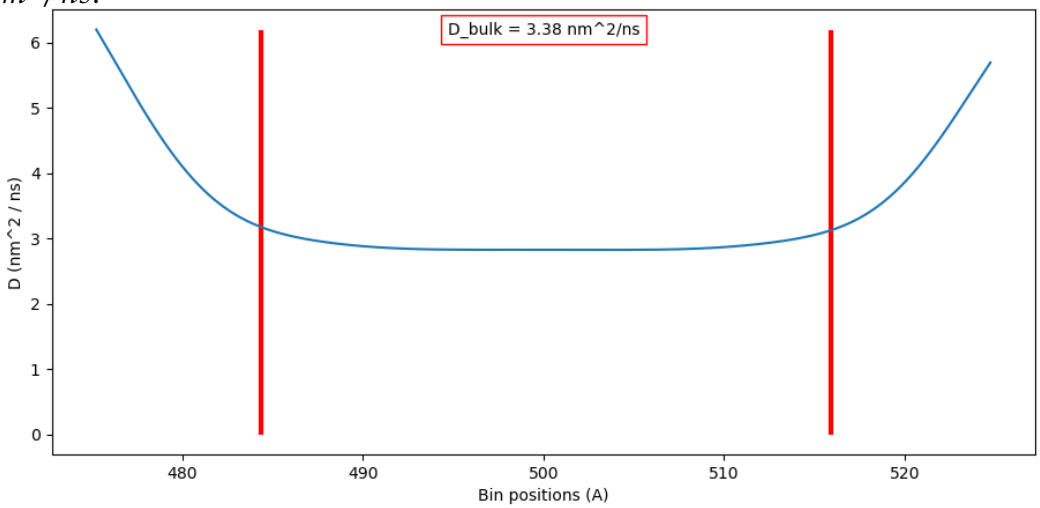
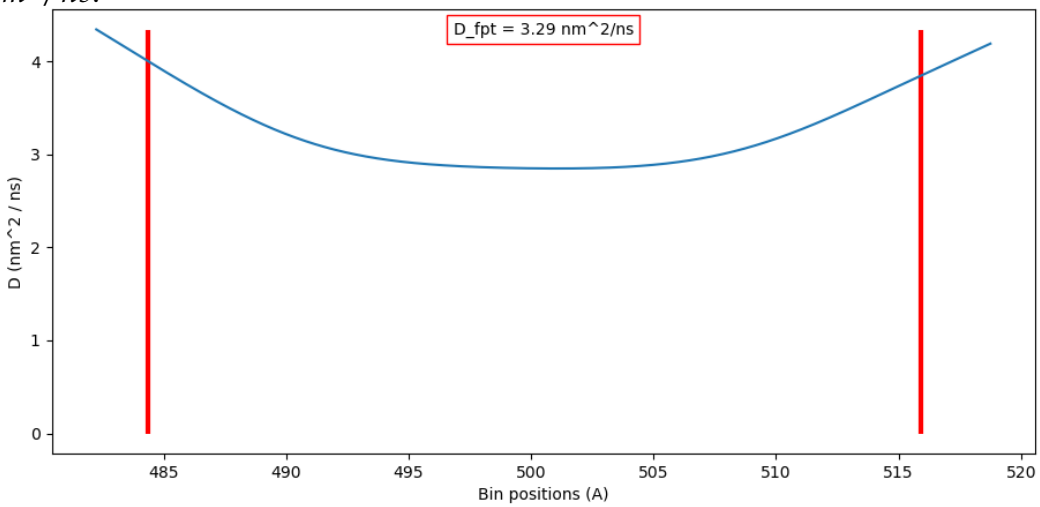


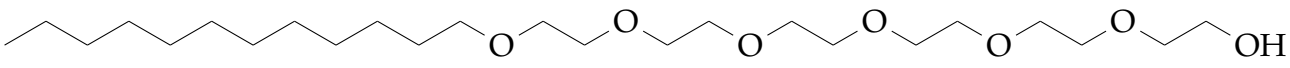
Figure 4.9: Estimating the space-dependent transverse diffusion coefficient from FP time statistics using Alg.2 for the water-air interface. The red bars represent the estimated Gibbs Dividing Interface (GDI). The blue line is drawn connecting the points that represent the D_z value in each bin. The estimated mean diffusion coefficient is $3.29 nm^2/ns$.



Finally, here we show the estimated diffusion coefficient as per the method of first-passage times in Fig.4.9 and as per Algorithm 5 in Fig.4.8. We expect the behaviour to be qualitatively and quantitatively similar between the two, if our theory is correct, since the method to produce Fig.4.8 is directly calculated from the trajectories itself.
And in fact, as we can see clearly, the 2 pictures are qualitatively similar in that the diffusion coefficient is constant in the bulk, and increases steeply as we near the interface on either side. This is because the water molecules are not favourably present in the region of air as the stabilising hydrogen bond networks between them are broken, and so they are more stable if they diffuse back quicker to the bulk. Hence, the diffusion coefficient increases as we near the interfaces.
Finally, the value of the diffusion coefficient in the bulk reported by the 2 methods are also similar - $3.38\, nm^2/ns$, $3.29\, nm^2/ns$, and this further justifies the validity of our analytical approximation of calculating the diffusion coefficient purely from FP-time statistics.

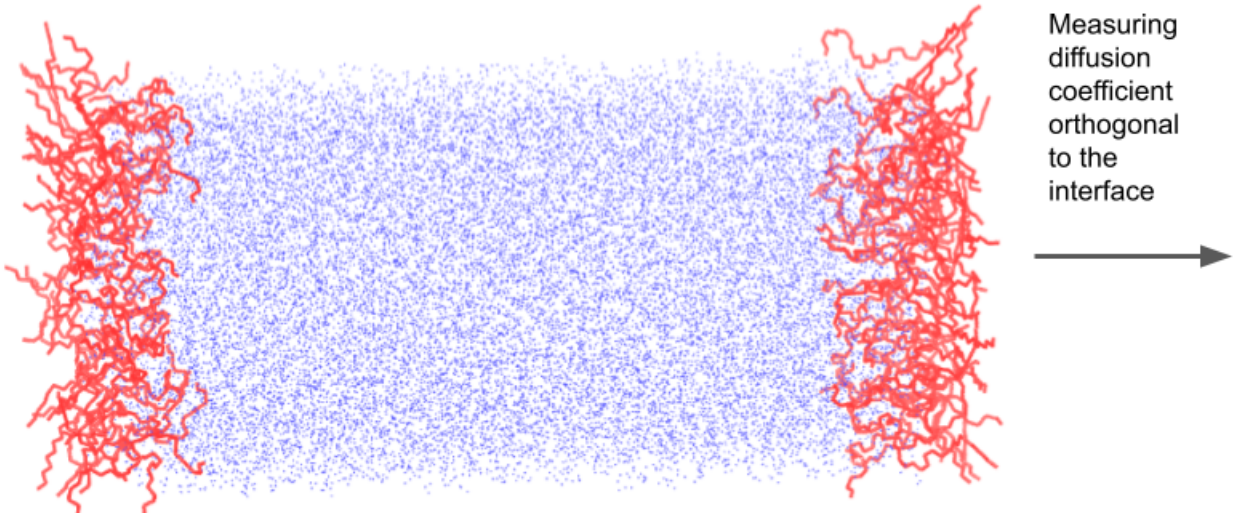
4.3 Water-C12E6 interface

In this section we reproduce results for a water-surfactant interface system from Ref.[2]. Our surfactant of choice is C12E6 (Hexaethylene Glycol Monododecyl Ether) and it has the following structure.



This is valuable to us since we have a clearly defined transition from a hydrophobic chain (-CH2-) to a hydrophilic component (ether groups). This helps us see some interesting, physically intuitive behaviour with respect to the diffusion coefficient.

Figure 4.10: Visualising the Water-C12E6 Interface. The water molecules are shown in blue, and the C12E6 molecules are shown in red. We measure the diffusion coefficient orthogonal to the water-C12E6 interface.



4.3.1 MD simulation details

For water, we use the SPC-E (Simple Point Charge Model with Effective Polarization Correction) as per Ref [9]. There are 8086 water molecules in a box of dimensions $[5.0, 5.0, 30.0]$ nm. There are 91 C12E6 molecules, 48 on the left of the water box, and 43 on the right.
We use the following parameters for the simulation.

- integrator = md (leap-frog algorithm)
- dt = 1 fs
- nsteps = 2 million
- total duration = 2 ns
- frequency of printing = 50 steps = 50 fs
- T = 298.15 K
- Thermostat = Velocity-Rescaling [10]
- Time constant for coupling = 0.1 ps
- PBC = X,Y,Z

For the FP analysis, we throw away the first 0.5 ns to ensure a well-equilibriated system.

4.3.2 FP analysis

This simulation is done for $dx = 0.5$ Å and $L = 1.0$ Å. In Fig.4.11, we see the mean first-passage time of water and as one would expect from a physical standpoint, the value in the bulk is similar to the value for Bulk Water for $L = 1.0$ Å.
The value of MFPT, in Fig.4.11, increases as we near the interface, and then drops away sharply as we transition from the hydrophilic to the hydrophobic component of the surfactant.
The value of the diffusion coefficient, in Fig.4.12 increases as we near the interface, and then drops away sharply as we transition from the hydrophilic to the hydrophobic component of the surfactant.
As we near the C12E6 interface, FP times start increasing. Water molecules first encounter the hydrophilic component of the surfactant molecules, and the stabilising interactions between the ether groups and water molecules slows down the diffusive dynamics of water in this region, and consequently, lead to an increase in the mean first-passage time, and decrease the diffusion coefficient.
Then, as we make a clear jump to the hydrophobic component of the surfactant molecules (due to distinctive nature of C12E6 we can cleanly define zones of hydrophobicity and hydrophilicity) we start to observe a sharp reduction in FP times. Now it is unfavourable for the water molecules to be present in the midst of non-polar alkyl groups due to the lack of stabilizing hydrogen-bond networks, and as a result, it favours diffusing out quickly from this region leading to a sharp decrease in the mean first-passage time, and an increase in the diffusion coefficient.

Figure 4.11: Estimating the mean first-passage time for the water-C12E6 interface. The black bars represent the estimated Gibbs Dividing Interface (GDI). The blue line is drawn connecting the points that represent the MFPT in each bin. In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å.

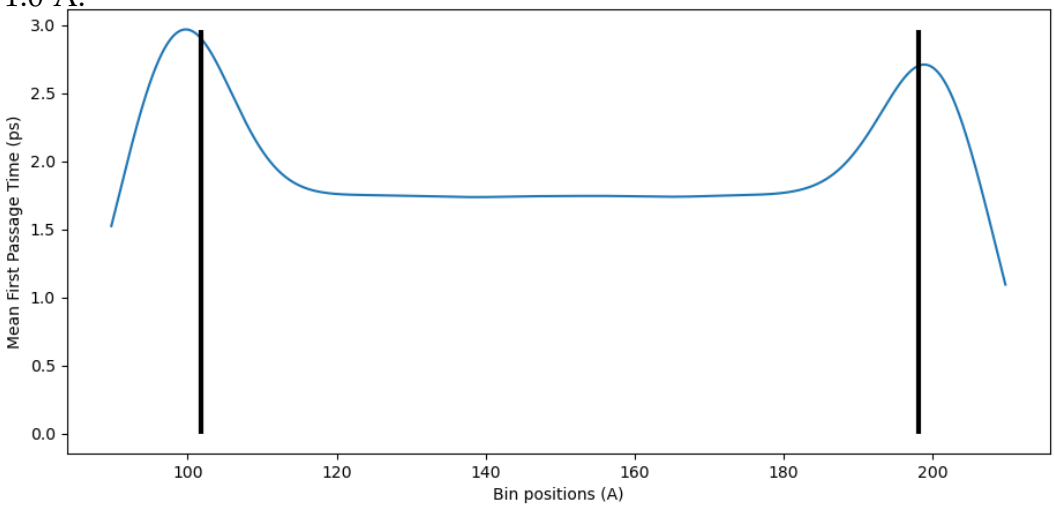
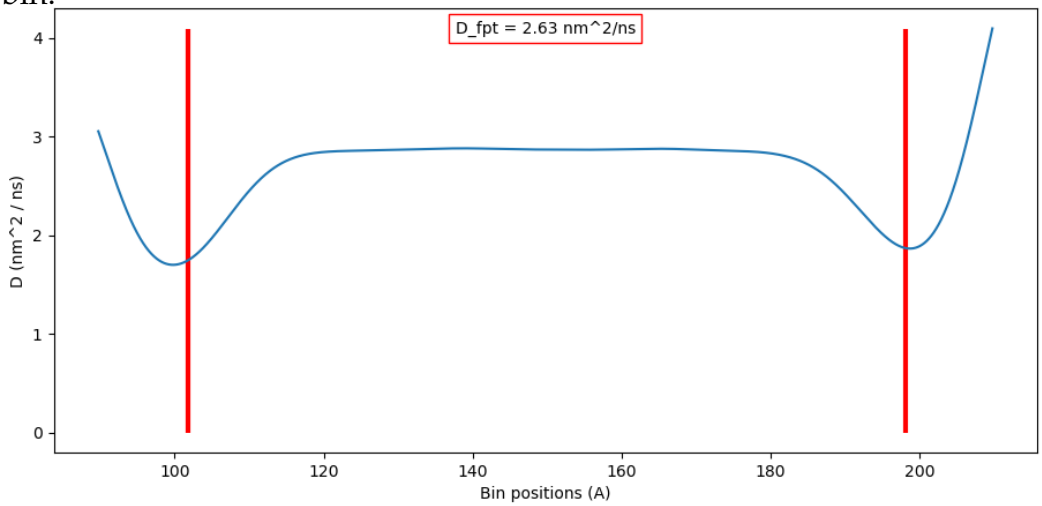


Figure 4.12: Estimating the space-dependent transverse diffusion coefficient for the water-C12E6 interface. The red bars represent the estimated Gibbs Dividing Interface (GDI). The blue line is drawn connecting the points that represent the D_z value in each bin.



4.3.3 Verification of Asymmetry in FP Analysis

As mentioned earlier, our system has 91 C12E6 molecules, 48 on the left of the water box, and 43 on the right. This asymmetry of distribution of surfactant molecules is very likely the cause of the asymmetry we observe between the crests in Fig.4.11 and the troughs in Fig.4.12.
To verify our hypothesis, we simply re-run the simulations, switching the asymmetry of the distribution of the molecules.

Figure 4.13: Estimating the MFPT of the flipped water-C12E6 interface. The black bars represent the estimated Gibbs Dividing Interface (GDI). Compare this to Fig.4.11.

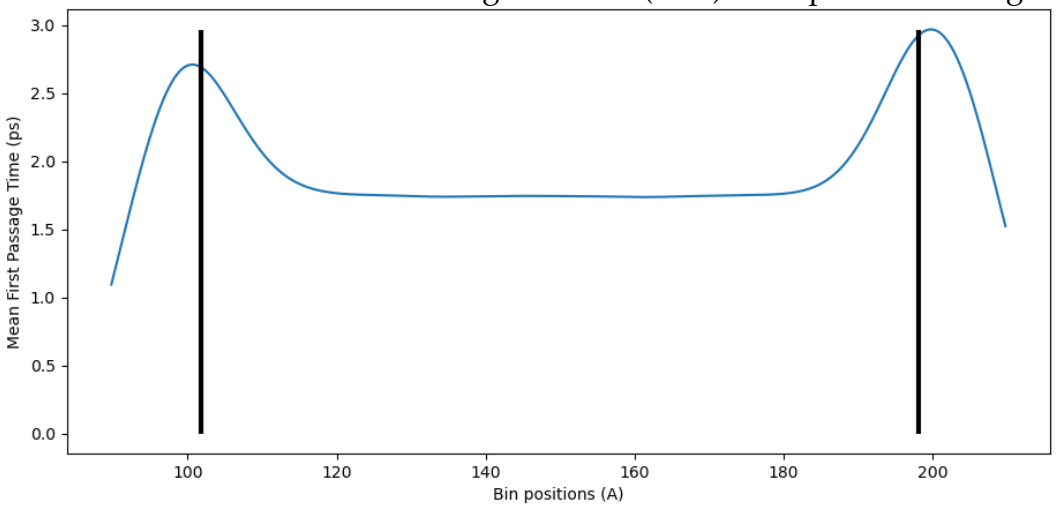
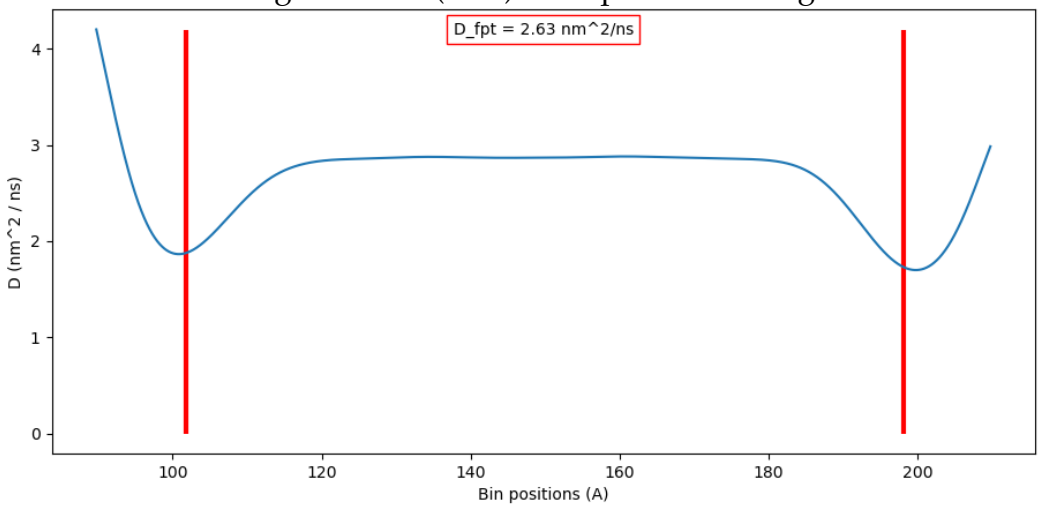


Figure 4.14: Estimating the space-dependent transverse diffusion coefficient of the flipped water-C12E6 interface from FP time statistics. The red bars represent the estimated Gibbs Dividing Interface (GDI). Compare this to Fig.4.12.

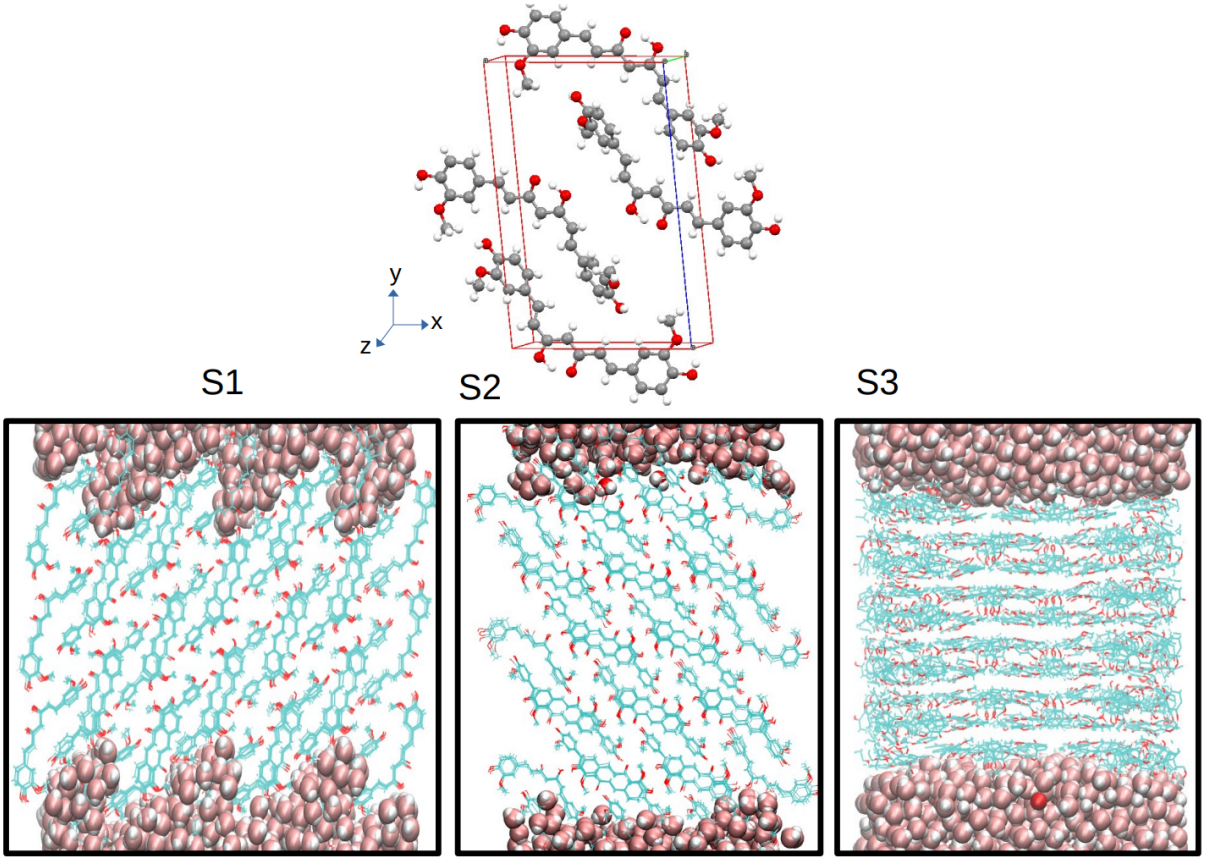


As one would expect, we notice that the higher crest has switched from the left interface in Fig.4.11 to the right interface in Fig.4.13. Similar behaviour is also observed with regard to the asymmetry in Fig.4.12 and Fig.4.14.

4.4 Water-Curcumin interface - Preliminary Results

Having established the accuracy of our toolbox for a variety of well-defined systems, we now start to explore a new system of interest without any ground truth. In Section 4.1, 4.2, 4.3, we have so far ensured that our toolbox can accurately reproduce the results of [1, 2]. Now, we explore the water-curcumin interface system.

Figure 4.15: The 3 different curcumin crystal surfaces - S1, S2, S3 that are used in the rest of this section are visualized here.



4.4.1 MD simulation details

The simulations were done by Dr. Daniel M Shadrack from the St. John's University of Tanzania.
We use the TIP4P water model. There are 180 curcumin molcules and 23 Sodium ions (Na^+) to try and create an electrically neutral system.
The number of water molecules and the exact box size varies for S1, S2, S3.

- S1: 6814 water molecules in a box of dimensions [13.50, 3.59, 5.91] nm. The curcumin crystal is solvated along the X axis.
- S2: 3502 water molecules in a box of dimensions [4.00, 3.50, 13.50] nm. The curcumin crystal is solvated along the Z axis.
- S3: 5821 water molecules in a box of dimensions [4.20, 13.50, 4.70] nm. The curcumin crystal is solvated along the Y axis.

We use the following parameters for the simulation of all 3 crystal surfaces.

- integrator = md (leap-frog algorithm)
- dt = 2 fs

- nsteps = 50 million
- total duration = 100 ns
- frequency of printing = 50 steps = 100 fs
- T = 300 K
- Thermostat = Velocity-Rescaling [10]
- Time constant for coupling = 0.1 ps
- PBC = X,Y,Z

For the FP analysis, we throw away the first 20 ns to ensure a well-equilibrated system.

4.4.2 S1 - FP analysis

This simulation is done for $dx = 0.5$ Å and $L = 1.0$ Å. In Fig.4.16 and Fig.4.19, we see the mean first-passage time of water and as one would expect from a physical standpoint, the value in the bulk is similar to the value for Bulk Water for $L = 1.0$ Å. The interesting behaviour happens in the middle of the spatial range where the curcumin crystal is present. As can be seen in Fig.4.16, for the time period of 21-30 ns, which is shortly after the system can be safely assumed to have reached equilibration, we see very asymmetric first-passage times in the crystal region, and there are sections of the crystal which show no first-passage events at all, that is, there is no water in these sections. The same features can be noticed in Fig.4.18, which shows no diffusive dynamics in the region of the curcumin crystal, and in Fig.4.17, which shows that water molecules coming close to the crystal (in the center) almost certainly tend to move away from it towards the bulk.

Figure 4.16: Estimating the mean first-passage time for the S1 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å. The discontinuity in the graph represents no recorded FP events, and therefore, no MFPT in a certain section of the curcumin crystal.

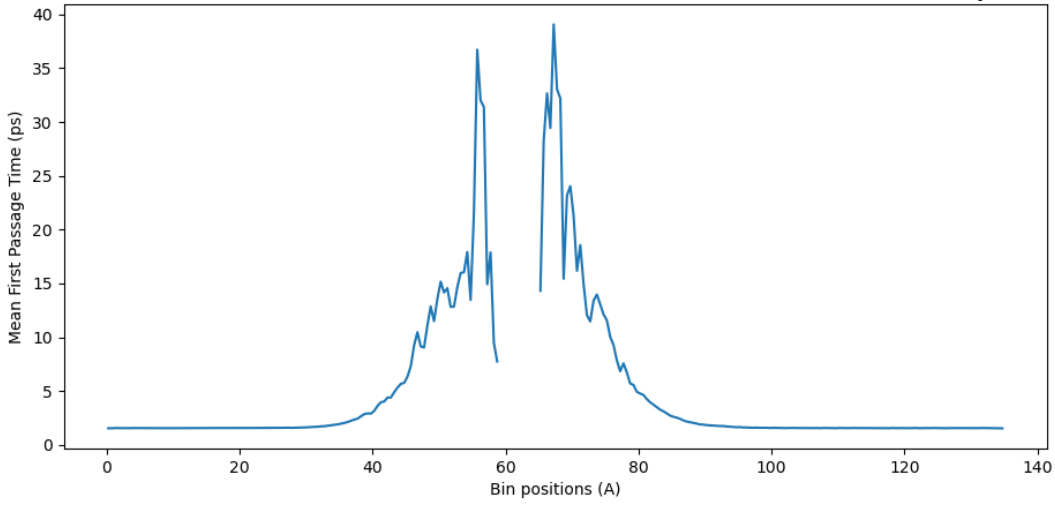


Figure 4.17: Comparing $P_+(\tau), P_-(\tau)$ for the S1 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The spikes represent very strong directional bias of diffusion of water molecules. The discontinuity in the graph represents no recorded FP events in a certain section of the curcumin crystal.

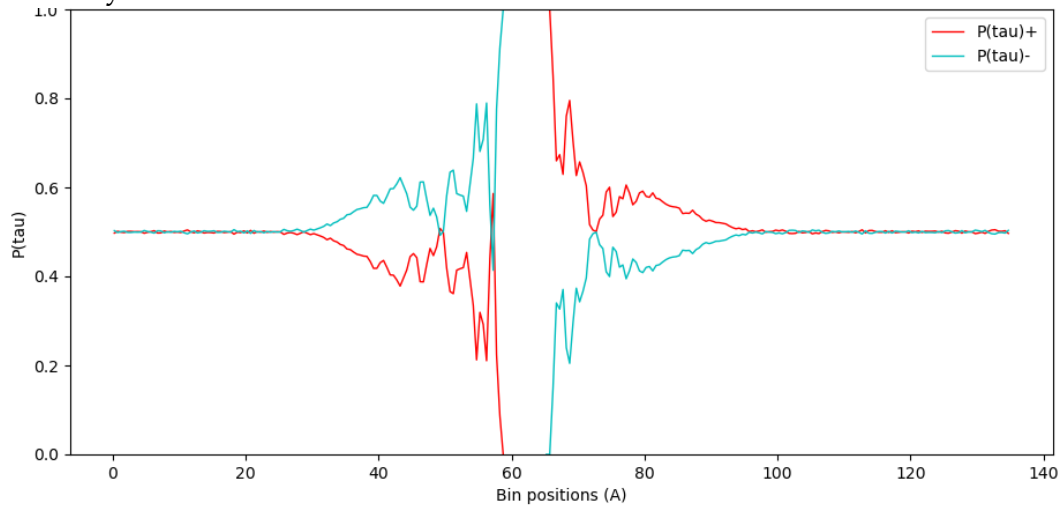
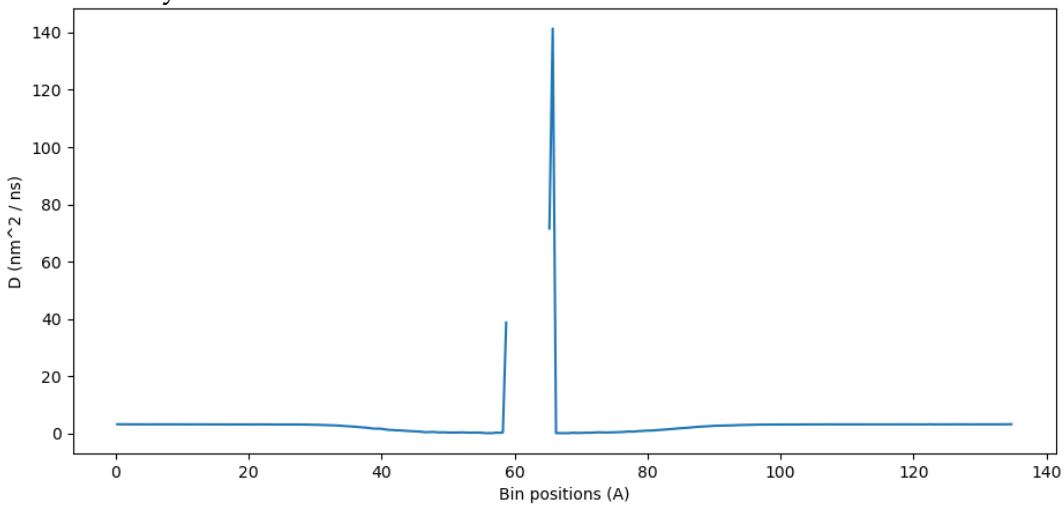


Figure 4.18: Estimating the space-dependent transverse diffusion coefficient for the S1 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns), from FP time statistics. The discontinuity in the graph represents no recorded FP events, and therefore, no diffusion in a certain section of the curcumin crystal.



Towards the end of the simulation, we can ascertain that there has been a small amount of water diffusion inside. In Fig.4.19 and Fig.4.21 , we see very slow, but constant diffusive dynamics (and high mean first-passage times) in the region of the crystal. Similarly, Fig.4.20 shows the equilibration of the probabilities in the crystal region, although water molecules at the edge of the crystal are still favoured to move towards the bulk.

In all cases the error bars for the mean value in each bin is smaller than the symbols used for plotting.

Figure 4.19: Estimating the mean first-passage time for the S1 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å.

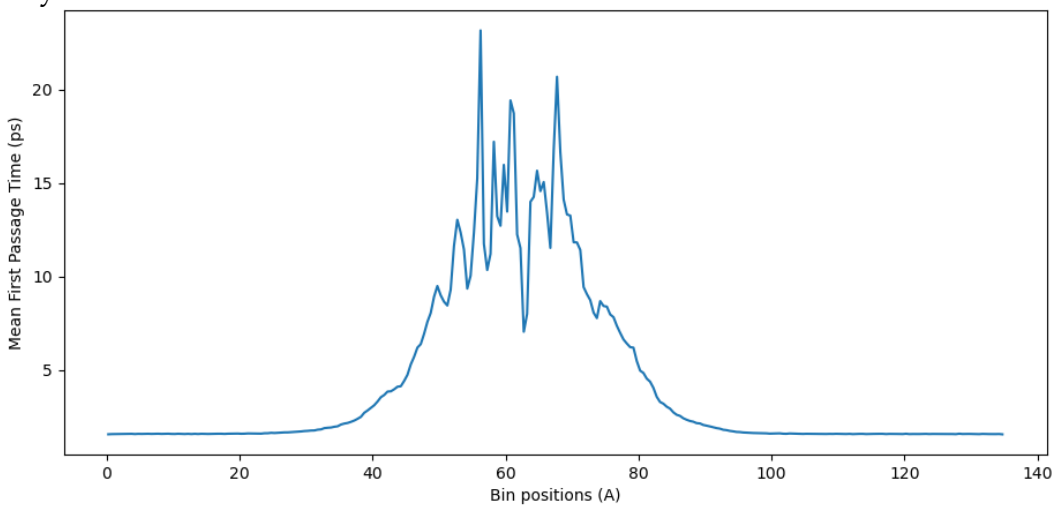


Figure 4.20: Comparing $P_+(\tau), P_-(\tau)$ for the S1 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The spikes represent strong directional bias of diffusion of water molecules.

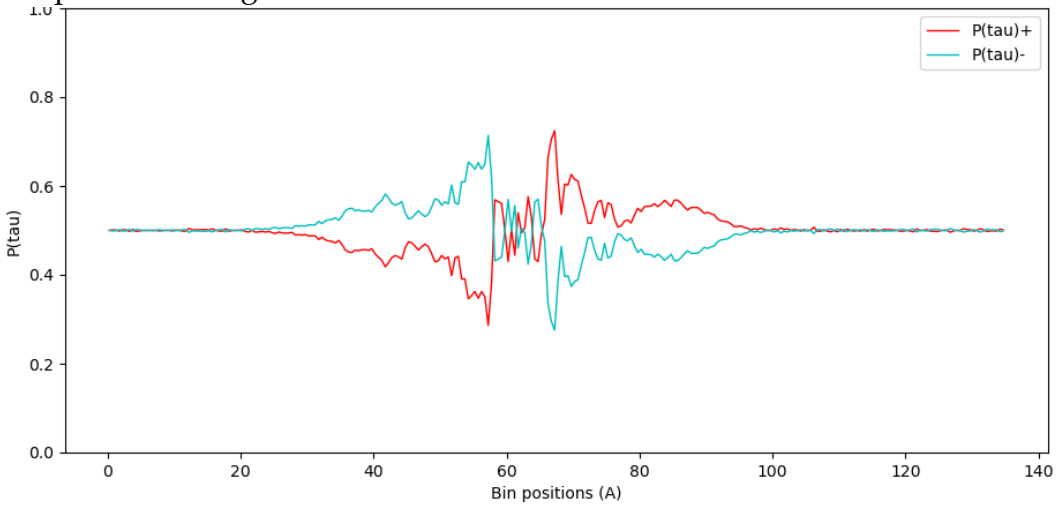
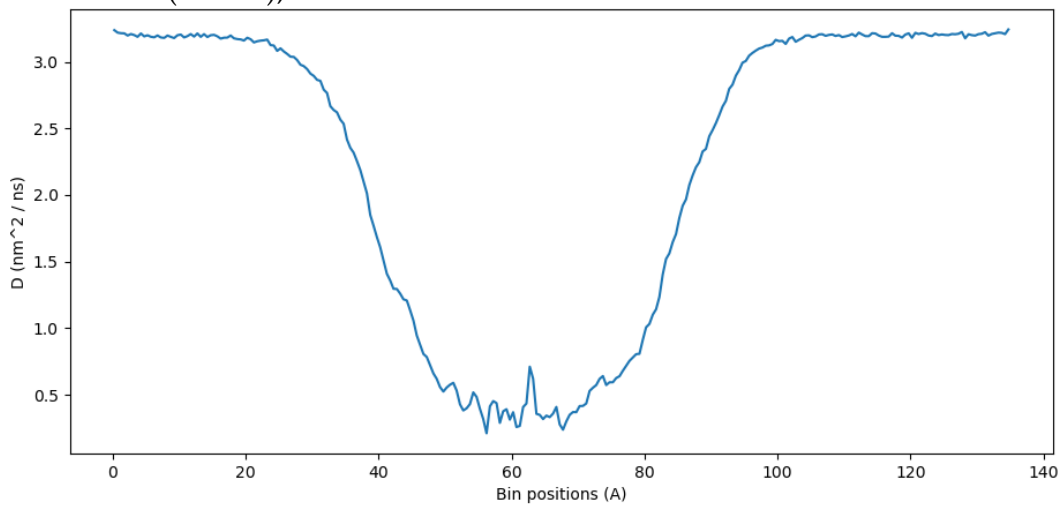
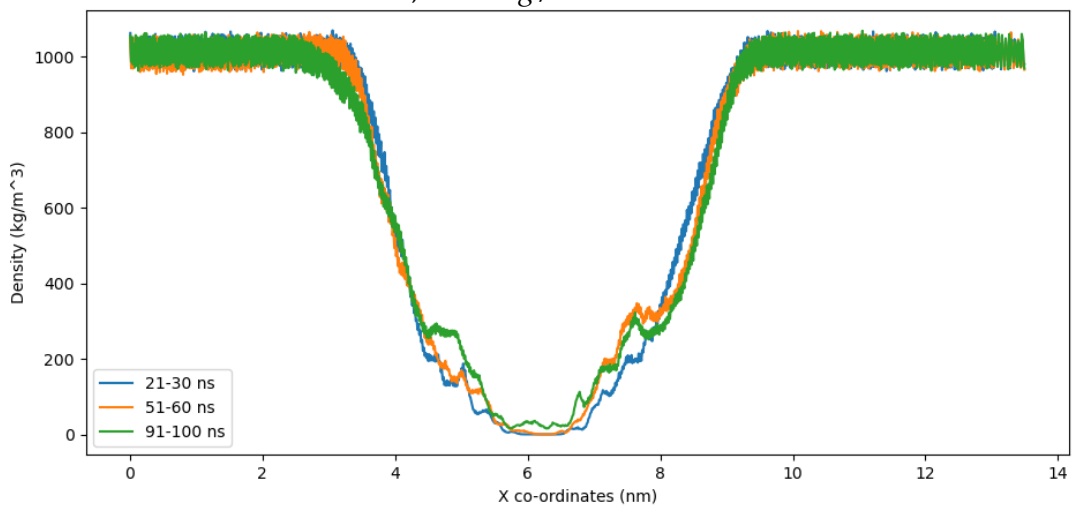


Figure 4.21: Estimating the space-dependent transverse diffusion coefficient for the S1 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns), from FP time statistics.



In Fig.4.22 we see the increase in the density of water in the region of the crystal from almost 0 at 21-30 ns, to 40 kg/m^3 for 91-100 ns.
This would seem to correlate well with our first-passage analysis.

Figure 4.22: Observing the evolution of water density for the S1 surface of the water-curcumin interface in the region of the curcumin crystal as the simulation progresses. The overall simulation duration is 100 ns, and we discard the first 20 ns to get a well-equilibrated system. We observe densities at 3 different time intervals: 21-30 ns, 51-60 ns, 91-100 ns. For S1, we notice an increase in the density of water in the region of the crystal from almost 0 at 21-30 ns, to 40 kg/m^3 for 91-100 ns.



4.4.3 S2 - FP analysis

This simulation is done for $dx = 0.5$ Å and $L = 1.0$ Å. In Fig.4.23 and Fig.4.26, we see the mean first-passage time of water and as one would expect from a physical standpoint, the value in the bulk is similar to the value for Bulk Water for $L = 1.0$ Å.

The S2 surface is particularly interesting, because unlike the small amount of diffusion in S1 or the larger degree of diffusion in S3, it shows almost no diffusion at all. In Fig.4.23, and Fig.4.25, we see that there is no diffusive behaviour at all in the region of the crystal.

Figure 4.23: Estimating the mean first-passage time for the S2 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å. The discontinuity in the graph represents no recorded FP events, and therefore, no MFPT in a certain section of the curcumin crystal.

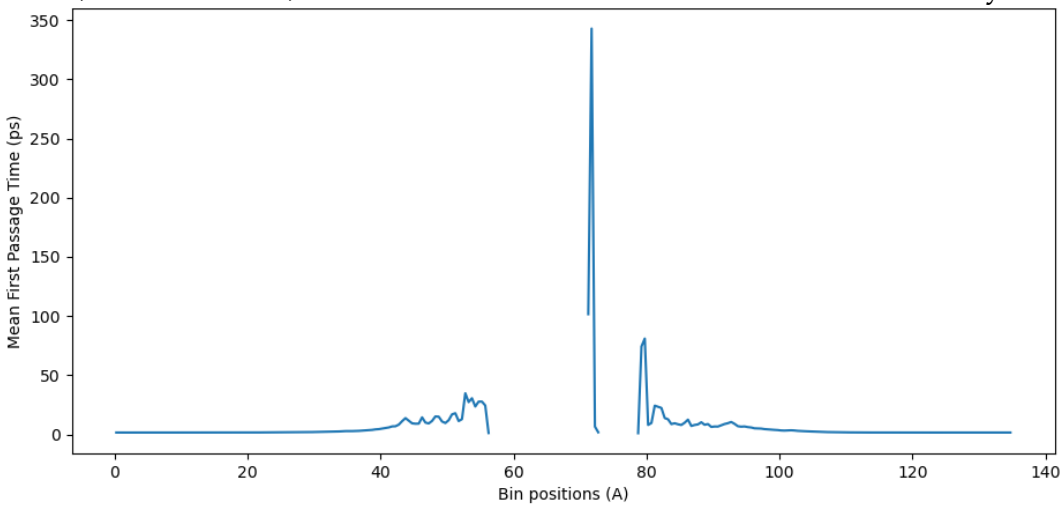


Figure 4.24: Comparing $P_+(\tau), P_-(\tau)$ for the S2 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The spikes represent very strong directional bias of diffusion of water molecules. The discontinuity in the graph represents no recorded FP events in a certain section of the curcumin crystal.

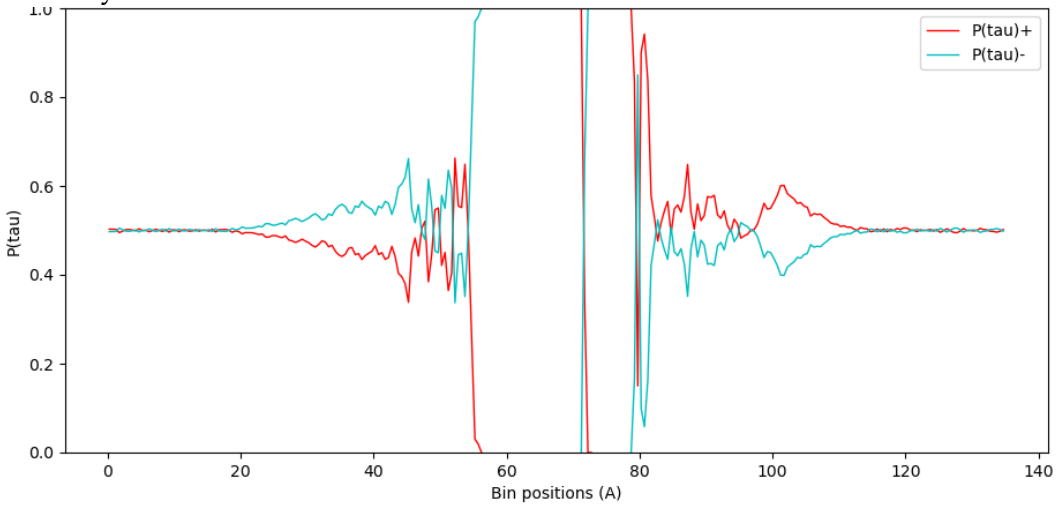
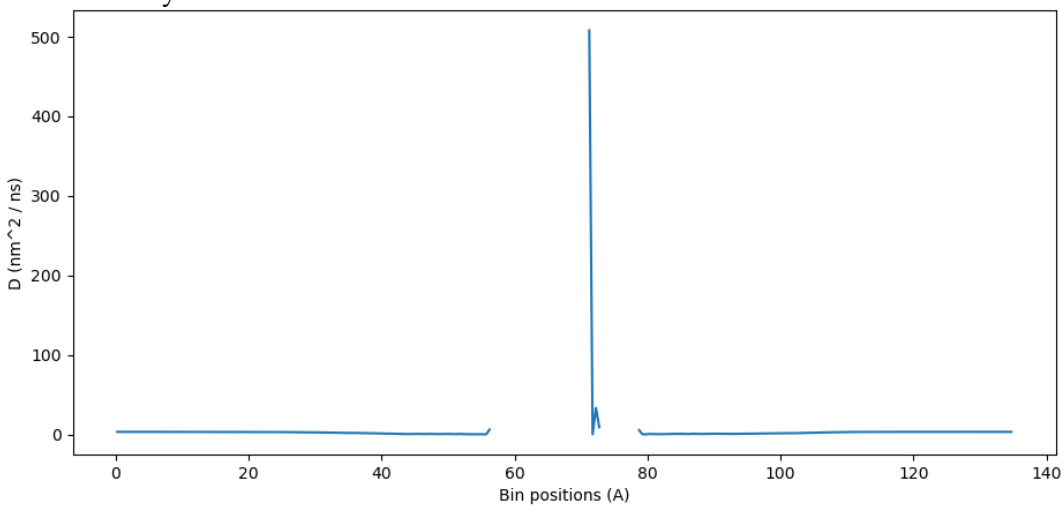


Figure 4.25: Estimating the space-dependent transverse diffusion coefficient for the S2 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns), from FP time statistics. The discontinuity in the graph represents no recorded FP events, and therefore, no diffusion in a certain section of the curcumin crystal.



There is no fundamental change to this behaviour even in the last 10 ns of the simulation. There has been a small degree of water diffusion, which leads to relatively more frequent (albeit very slow) first-passage events, but there are still sections of the crystal which show no diffusive dynamics at all, suggesting that water has failed to penetrate it.
In all cases the error bars for the mean value in each bin is smaller than the symbols used for plotting.

Figure 4.26: Estimating the mean first-passage time for the S2 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å. The discontinuity in the graph represents no recorded FP events, and therefore, no MFPT in a certain section of the curcumin crystal.

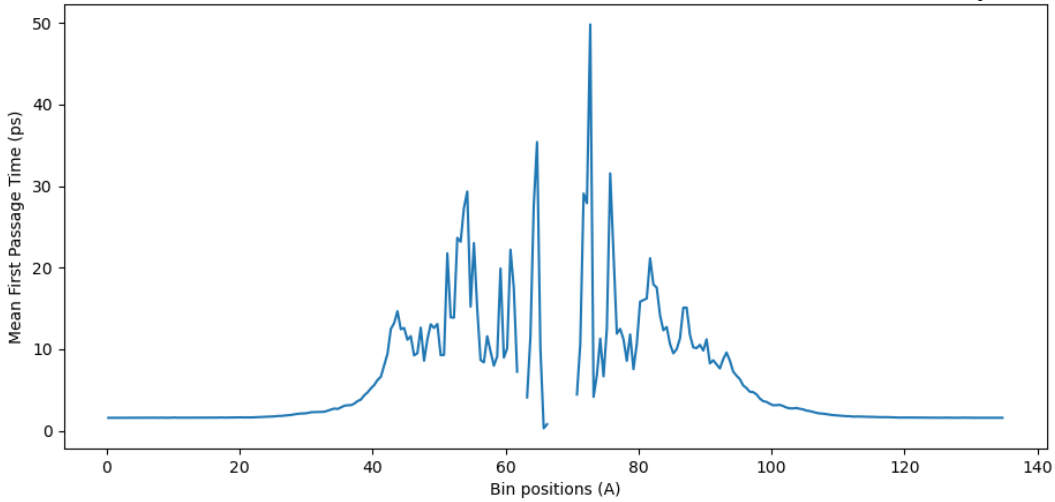


Figure 4.27: Comparing $P_+(\tau), P_-(\tau)$ for the S2 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The spikes represent very strong directional bias of diffusion of water molecules. The discontinuity in the graph represents no recorded FP events in a certain section of the curcumin crystal.

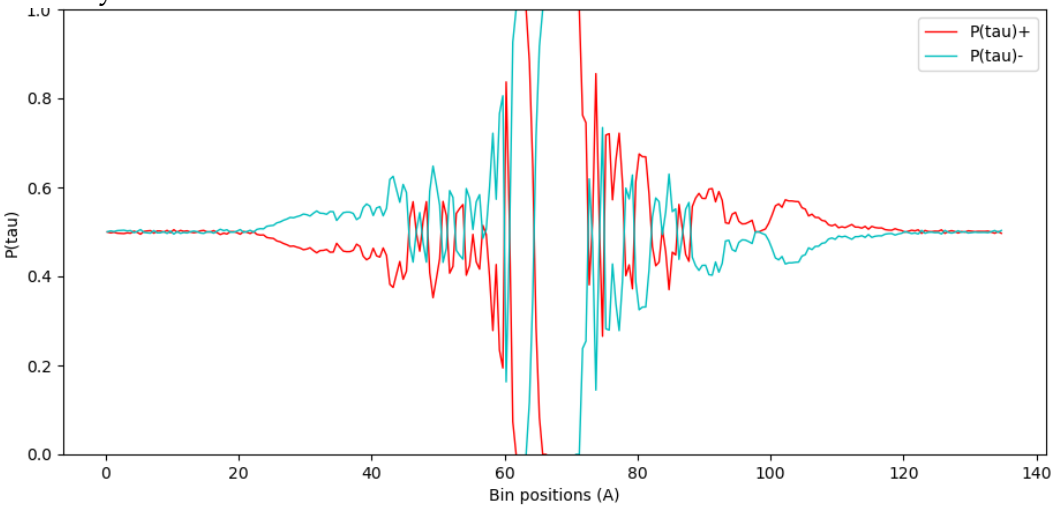
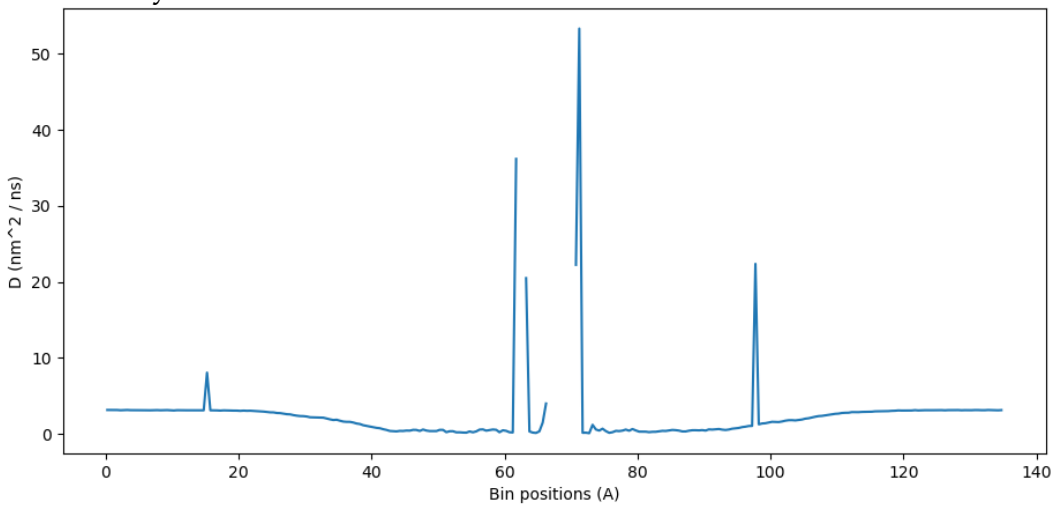
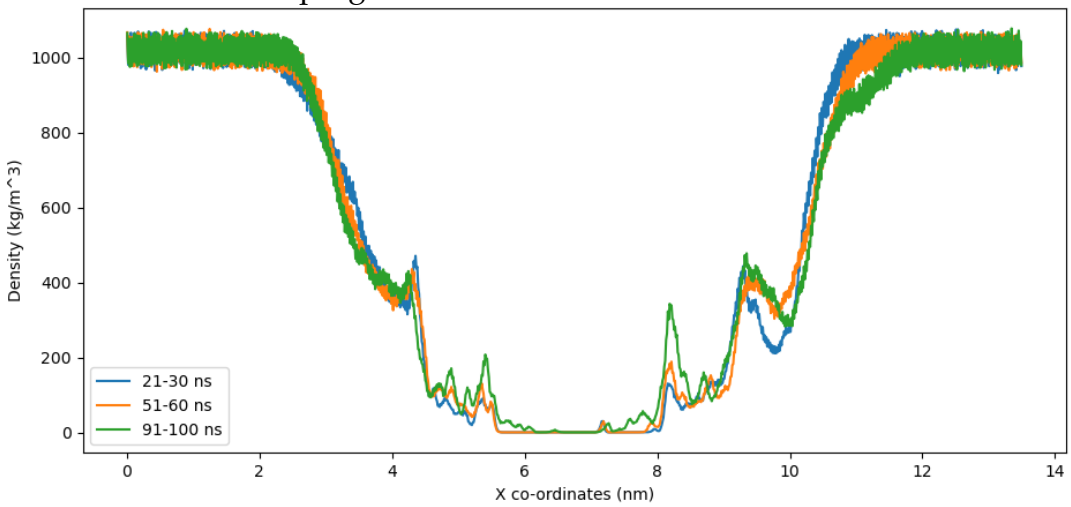


Figure 4.28: Estimating the space-dependent transverse diffusion coefficient for the S2 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns), from FP time statistics. The discontinuity in the graph represents no recorded FP events, and therefore, no diffusion in a certain section of the curcumin crystal.



In Fig.4.29 we see that the density of water in the region of the crystal barely shifts over the 100 ns simulation. This would seem to correlate well with our first-passage analysis.

Figure 4.29: Observing the evolution of water density for the S2 surface of the water-curcumin interface in the region of the curcumin crystal as the simulation progresses. The overall simulation duration is 100 ns, and we discard the first 20 ns to get a well-equilibrated system. We observe densities at 3 different time intervals: 21-30 ns, 51-60 ns, 91-100 ns. For S2, we clearly see almost no diffusion of water inside the crystal region as the simulation progresses.



4.4.4 S3 - FP analysis

This simulation is done for $dx = 0.5$ Å and $L = 1.0$ Å. In Fig.4.30 and Fig.4.33, we see the mean first-passage time of water and as one would expect from a physical standpoint, the value in the bulk is similar to the value for Bulk Water for $L = 1.0$ Å. The interesting behaviour happens in the middle of the spatial range where the curcumin crystal is present. For the time period of 21-30 ns, which is shortly after the system can be safely assumed to have reached equilibration, we see very asymmetric first-passage times in the crystal region. There are some sharp spikes associated with very slow diffusive dynamics all the way up to 30 ps. In Fig.4.31 we see very sharp spikes in directional probabilities all the way up to 0.9 or down to 0.1. In Fig.4.32, we notice very asymmetric diffusive dynamics in the region of the crystal.

Figure 4.30: Estimating the mean first-passage time for the S3 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å.

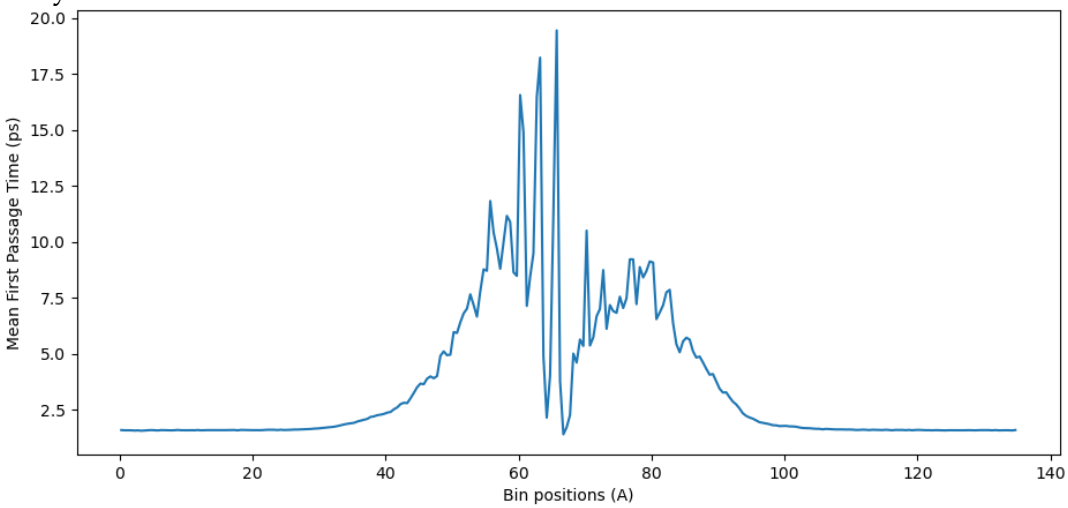


Figure 4.31: Comparing $P_+(\tau), P_-(\tau)$ for the S3 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The spikes represent very strong directional bias of diffusion of water molecules.

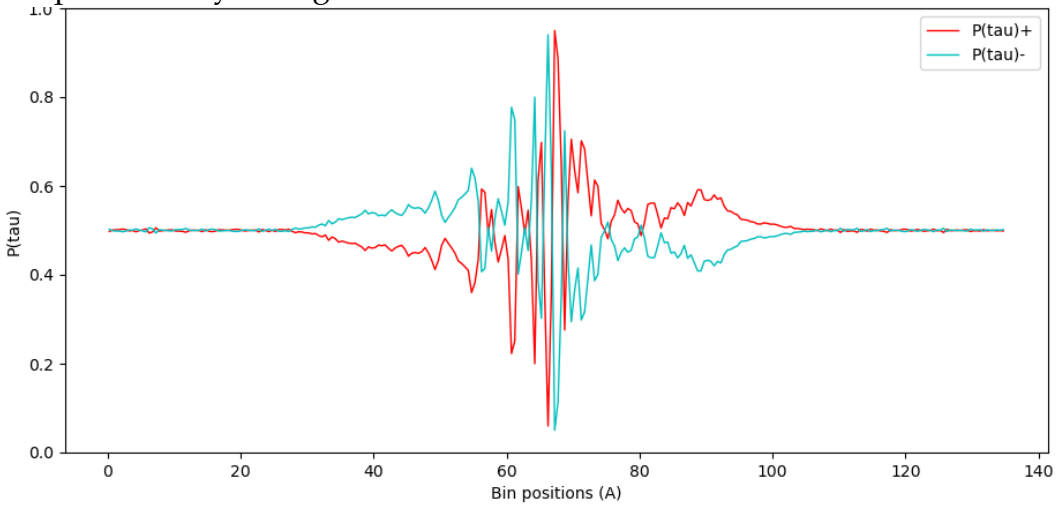
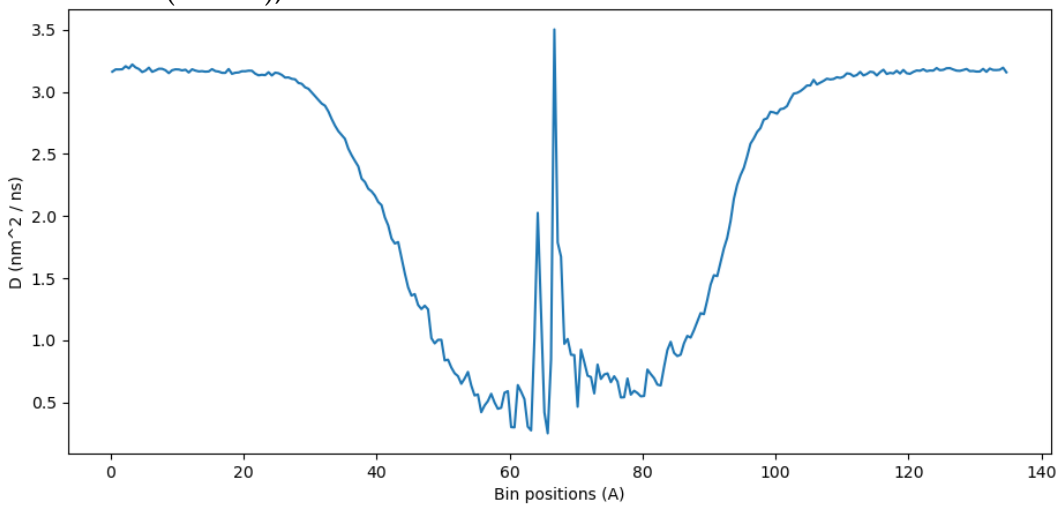


Figure 4.32: Estimating the space-dependent transverse diffusion coefficient for the S3 surface of the water-curcumin interface for the range of 21-30 ns of the overall simulation time (100 ns), from FP time statistics.



As we look towards the end of the simulation, the region from 90-100 ns, we now see a stabilisation of the mean first-passage times in the region of the crystal to around 10-12 ps. In Fig.4.34 we also see directional probabilities have now calmed down, and the range of spikes is between 0.6 to 0.4. In Fig.4.35, we notice a much more gradual (albeit slow) diffusive behaviour of water.
In all cases the error bars for the mean value in each bin is smaller than the symbols used for plotting.

Figure 4.33: Estimating the mean first-passage time for the S3 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, the MFPT $\approx 1.7ps$ which is consistent with the values for the bulk water system when $L = 1.0$ Å.

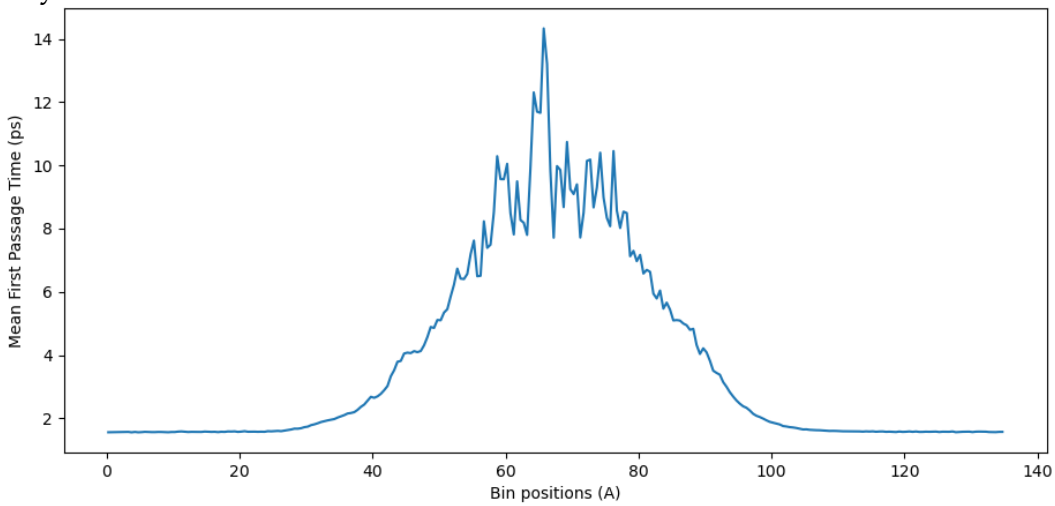


Figure 4.34: Comparing $P_+(\tau), P_-(\tau)$ for the S3 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns). In the bulk region, $P_+(\tau) \approx P_-(\tau) \approx 0.5$ which is consistent with the expected behaviour. The increased water diffusion in S3 shows a significantly smoother probability curve than in S1 or S2, with spikes limited between 0.4-0.6

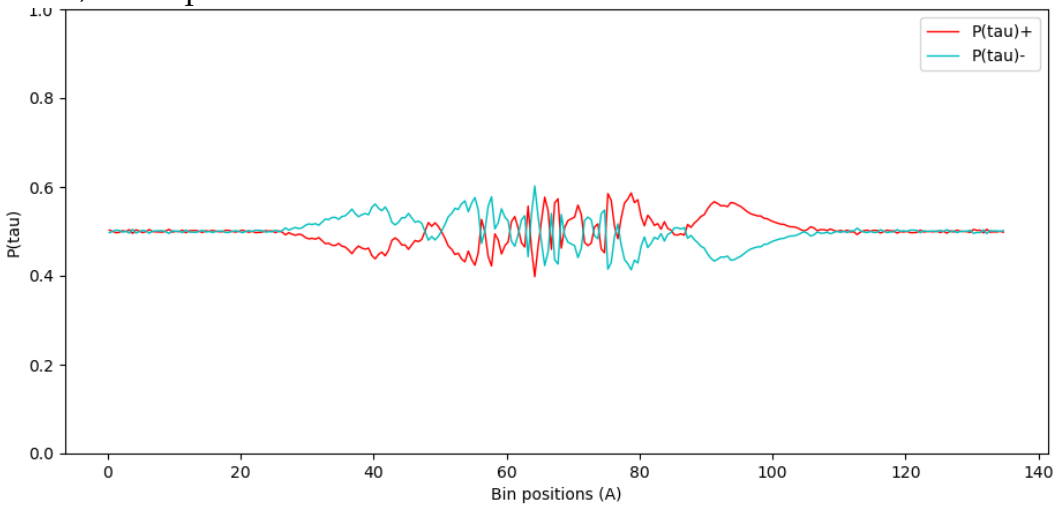
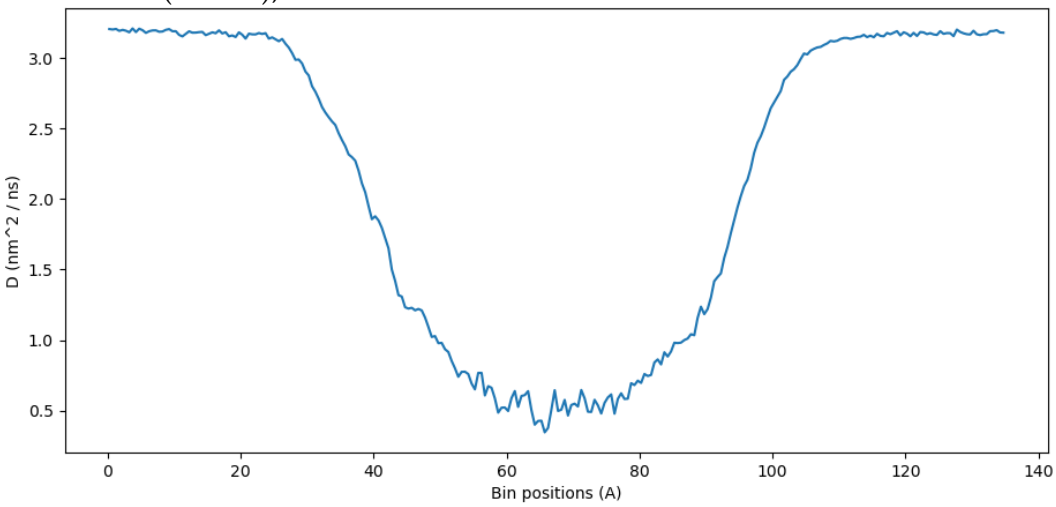


Figure 4.35: Estimating the space-dependent transverse diffusion coefficient for the S3 surface of the water-curcumin interface for the range of 91-100 ns of the overall simulation time (100 ns), from FP time statistics.



Physically, one can interpret these to suggest that significant water diffusion into the crystal has occurred and it has reached a point of stability, such that now even though the diffusive dynamics are slower in the crystal, they are mostly equi-probable from a directional perspective, and have a well-defined mean first-passage time. This is different from the S1 case where the edges of the crystal still showed significant directional probability bias. Most probably this difference can be explained by the overall increased degree of water diffusion inside the crystal region.
The final justification of our toolbox's ability to explore a previously undefined system is the comparison of our hypotheses to the evolution of the density profile of water as obtained from GROMACS.

In Fig.4.36 we see the increase in the density of water in the region of the crystal from almost 0 at 21-30 ns, to 170 kg/m^3 for 91-100 ns.
This would seem to correlate well with our first-passage analysis.

Figure 4.36: Observing the evolution of water density for the S3 surface of the water-curcumin interface in the region of the curcumin crystal as the simulation progresses. The overall simulation duration is 100 ns, and we discard the first 20 ns to get a well-equilibrated system. We observe densities at 3 different time intervals: 21-30 ns, 51-60 ns, 91-100 ns. For S3, we clearly see the increase in the density of water in the region of the crystal from almost 0 at 21-30 ns, to 170 kg/m^3 for 91-100 ns.

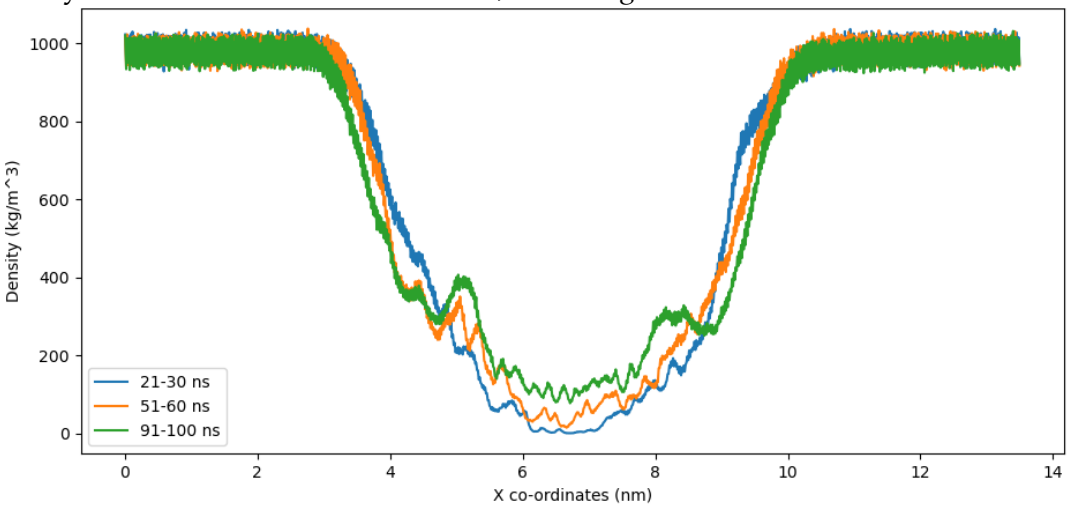
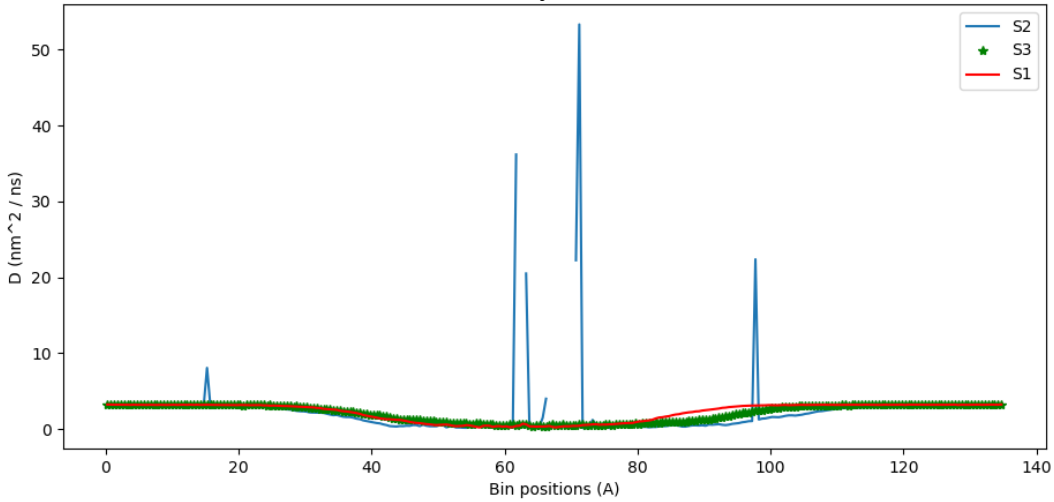


Figure 4.37: The comparison of the space-dependent transverse diffusion coefficients inferred from first-passage statistics at the end of the simulation (91-100 ns) show the discrepancy when water diffuses inside the crystal slightly (S1), or significantly (S3), versus when it doesn't diffuse inside the crystal at all (S2).



Chapter 5

Parallelisation and Performance Analysis

5.1 Overview

In CPython, the global interpreter lock (GIL), is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once. The GIL prevents race conditions and ensures thread safety.
In simple words, it is a lock that allows only one thread to hold the control of the Python interpreter.
This means that only one thread can be in a state of execution at any point in time. The impact of the GIL isn't visible when executing a single-threaded program, but it can be a performance bottleneck in CPU-bound and multi-threaded code.
In PyFPT, we aim to use parallelism to improve the performance of our code, since some aspects of the analysis that we undertake can be considered to be embarrassingly parallel. To this end, we use a multi-processing approach where you use multiple processes instead of concurrent threads. Each Python process gets its own Python interpreter and memory space so the GIL won't be a problem. Each such process can run on separate CPU cores (as long as they are limited to the number of available cores).
We use Dask [12] for parallelising our code in Python3 in the multiprocessing framework.
Dask creates multiple Python processes, that can be executed either as soon as they are created (`futures`) or lazily (`dask.delayed`) as the requirement arises. This doesn't actually change the time it takes to execute the function itself, but is more a matter of load-balancing for complicated task graph scenarios.
In our implementation, we have both of these options but since the processes are run concurrently, the lazy execution approach is not the default.
The primary bottleneck of this approach of parallelism is that since each process has its own memory space, we become memory-bound in our computational capabilities if we use Algorithm 1 in a naive fashion as we keep loading the trajectory into memory for every available process. To prevent this problem, our function to read the data into memory only loads the trajectories for those atoms that are processed by a certain parallel worker, and hence, we at least prevent the worst effects of this bottleneck.
Algorithm 2 becomes necessary for the scenario where enough system memory is not available, particularly for users who don't have access to a high-performance compute

cluster.

5.2 Parallelising PyFPT

There are 2 major opportunities for parallelising.

- First-Level - Atom-Parallelisation:The first is in estimating the mean first-passage time using either Algorithm 1 or 2 over a fixed range of the trajectory (or even the entire trajectory). Here, we can parallelise over the total number of atoms. For example, if our system has 100 atoms and we have 4 CPU cores, each core can process the full trajectory of 25 atoms. This is the simplest and most obvious level of parallelism.

  This is implemented (in the `futures` method) as follows

```
def concurrent_parallel_fpt(...):
...
    client = Client(n_workers=nprocs, memory_limit="params.sys_mem_limit")

    futures = []

    # Parallelizing over atoms
    # Keep a large atom_step
    for at_id in range(1, atom_step + 1):
        dt,box_dims,coord_data_local = read_traj_ndarray(
                atom_group,
                frame_params_list,
                at_id,
                atom_end,
                atom_step,
            )

        futures.append(
            client.submit(
                fpt,
                coord_data_local,
                frame_params_list,
                at_id,
                atom_end,
                atom_step,
                box_dims,
                dt,
                L,
                bin_params_list,
                params.correlation_time_val,
                ax_id_bin,
                ax_id_fpt,
            )
        )
```

```
    result_new = client.gather(futures)
...
```

This results in parallelisation over the number of atoms by modifying a parameter `atom_start`, which makes each process start from a different atom, skip over `atom_step` number of atoms, and process their trajectories all the way until it hits the last atom in the sequence.

`atom_start` can only take values from 1 to `atom_step`, so the combination of these 2 parameters control our parallelisation.

For the sake of memory efficiency, we also only load into each process the position coordinates of atoms that the specific process will be operating on. Otherwise each process would have the full trajectory in its local memory space and that would quickly make our approach untenable.

The `atom_step` parameter is carefully controlled to be the same as number of processes being used for parallelisation. However, since this toolbox is meant to be used by people without much programming knowledge, there are automated checks and corrections in place to fix this parameter if an unreasonable demand is made.

```
# n_proc -> number of CPUs user asks for
# n_cpus -> total number of available CPUs
if (n_proc > atom_step) and (n_proc <= n_cpus):
    atom_step = n_proc
elif (n_proc > atom_step) and (n_proc >= n_cpus):
    atom_step = n_cpus
    n_proc = n_cpus
elif (atom_step >= n_cpus) and (n_proc <= n_cpus):
    atom_step = n_proc
elif (atom_step >= n_cpus) and (n_proc >= n_cpus):
    atom_step = n_cpus
    n_proc = n_cpus
elif (atom_step < n_cpus) and (n_proc < atom_step):
    atom_step = n_proc
```

- Second-Level - Time-Parallelisation: The second is parallelising over time. This can potentially be somewhat scientifically unsound and risky and is turned off by default, and doesn't make sense for short trajectories of a few nanoseconds. This is because it may lead to us not correctly counting or attributing the mean first-passage time itself, since we are breaking up the trajectory over time. However, there are potential use-cases for this level of parallelism as well. This is useful for dynamically evolving systems (such as our water-curcumin example), where we are interested in the evolution of the mean first-passage times and associated diffusive dynamics in certain pre-specified time intervals. In the water-curcumin system, for a 100 ns trajectory, we split it up in blocks of 10 ns.

  This is implemented (in the `futures` method) as follows

```
def parallel_time(...):
...
    # n_traj_divs -> number of divisions of full trajectory

    frame_params_list = [
        params.frame_start,
        params.frame_end,
        params.frame_check_interval,
    ]

    n_frames = params.frame_end - params.frame_start

    n_frames_div = int(n_frames/n_traj_divs)

    client = Client(n_workers=nprocs, memory_limit="params.sys_mem_limit")

    job_list = []

    for traj_id in range(0, n_traj_divs-1):

        frame_start = params.frame_start + (traj_id * n_frames_div)

        frame_end = (traj_id + 1) * n_frames_div

        new_frame_params_list = [
            frame_start,
            frame_end,
            params.frame_check_interval,
        ]

        job_list.append(
            dask.delayed(concurrent_parallel_fpt)(
                n_procs,
                new_frame_params_list,
                atom_end,
                atom_step,
                box_dims,
                dt,
                L,
                bin_params_list,
                ax_id_bin,
                ax_id_fpt,
            )
        )

    results = dask.compute(job_list)
```

This results in parallelisation over chunks of trajectory. As mentioned earlier, one must be exceedingly careful when doing this.

Here, we are breaking up the trajectory into a number of chunks as defined by the user (default is set to 1, so no breakups are done), then we modify the frame starting and ending ranges for each of these chunks, and create corresponding jobs to do the FPT analysis for each of these chunks. That is, each of the `dask` jobs created here itself calls the `concurrent_parallel_fpt()` function described in the previous point, and then they all launch the required number of workers to parallelise over atoms. Thus, we have 2 levels of parallelisation.

Also, here we use the `dask.delayed` interface instead of the `futures` interface, since we want to ensure that every job (call to `concurrent_parallel_fpt`) is created prior to execution. This is recommended in the best practices for creating nested dask jobs.

To time the function execution times, we use the following code as a function decorator as shown below.

```
from timeit import default_timer as timer

def func_timer(func):
    def wrap_func(*args, **kwargs):
        t1 = timer()
        result = func(*args, **kwargs)
        t2 = timer()
        print(f"\nFunction {func.__name__!r} executed in {(t2-t1):.4f}s\n")
        return result

    return wrap_func


@func_timer
def fpt():
...
```

5.3 System Architecture

All the tests were run on MARCONI-100 cluster. MARCONI-100 is the new accelerated cluster based on IBM Power9 architecture and Volta NVIDIA GPUs, acquired by Cineca within PPI4HPC European initiative.
It has the following composition

- Nodes: 980
- Processors: 2x16 cores IBM POWER9 AC922 at 3.1 GHz
- Accelerators: 4 x NVIDIA Volta V100 GPUs, Nvlink 2.0, 16GB
- Cores: 32 cores/node
- RAM: 256 GB/node
- Peak Performance: 32 PFlop/s

Theoretical Peak Performance per node	
CPU (nominal/peak freq.)	691/791 GFlops
GPU	31.2 TFlops
Total	32 TFlops
Memory Bandwidth (nominal/peak freq.)	220/300 GB/s

- OS: RedHat Enterprise Linux 8.1

Our program is only using the CPUs, so the GPU performance is not particularly relevant for our use-case.
However, when doing the MD simulations that serve as input to our software, we use the GPUs which serve as massively beneficial accelerators.
Some details about the CPUs obtained using `lscpu`.

```
Architecture:        ppc64le
Byte Order:          Little Endian
CPU(s):              128
On-line CPU(s) list: 0-127
Thread(s) per core:  4
Core(s) per socket:  16
Socket(s):           2
NUMA node(s):        6
Model:               2.1 (pvr 004e 1201)
Model name:          POWER9, altivec supported
CPU max MHz:         3800.0000
CPU min MHz:         2300.0000
L1d cache:           32K
L1i cache:           32K
L2 cache:            512K
L3 cache:            10240K
NUMA node0 CPU(s):   0-63
NUMA node8 CPU(s):   64-127
NUMA node252 CPU(s):
NUMA node253 CPU(s):
NUMA node254 CPU(s):
NUMA node255 CPU(s):
```

5.4 Performance Comparison to Existing Methodologies

The code that we have developed has an incredible performance improvement, even in the serial version, over the previous method that the group had. This previous method was in rudimentary Python and used some Mathematica to tie up the ends. It had no parallelisation support.
If we look at even just the serial version, we see almost 2 orders of magnitude improvement in performance, and when we focus on even moderate parallelisation (4-core

Performance Comparison for Bulk Water System	
Original Code	19000 sec
Our version (1 core)	220 sec
Performance Improvement for Serial Version	86x ($\sim 10^2$)
Our version (4 cores)	60 sec
Performance Improvement for 4-core Version	3100x ($\sim 10^3$)
Our version (128 cores - 1 node on Marconi-100)	2 sec
Performance Improvement for 128-core Version	9500x ($\sim 10^4$)

systems, as can be found on many modern laptops), we have 3 orders of magnitude improvement in performance.
Finally, when we compare the original methodology to the full performance we can obtain on 1 node of Marconi-100, we have a nearly 4 orders of magnitude improvement in performance.
This is a real testament to the massive improvements that can be observed by reading the trajectory into memory, designing an efficient algorithm, and taking advantage of "embarrassingly parallel" sections of a problem.

5.5 Scaling

Here we present the result of the scaling of the `fpt()` function. It is important to note that the parallelisation over atoms that is done with the `futures` interface of dask comes with a 1 ms overhead for each task. The parallelisation over time which is optional has a 200 μs overhead for each task.
We also present the results for scaling for multiple threads/processes on a single node of Marconi-100. This is due to the nature of the code, which is not implemented (and was not intended) as a distributed memory program.
If one doesn't parallelise over time, then the scaling of the `fpt()` function is simply measured over the total number of atoms being distributed among the available CPU threads.
If one choose to parallelise over time, then it is important to be vigilant that

```
max (num_divisions_time * atom_step) = num_threads
```

where `num_divisions_time` is the number of splits of the trajectory over time, `atom_step` is the number of atoms each process skips over (analogous to the total number of threads processing that section of the trajectory for all atoms), `num_threads` is the total number of threads available in the system (with hyperthreading enabled).
If we request more overall threads than the system has available then we will actually slow down the performance since some processes will all be run simultaneously, but not using the full single-core performance of the CPU, and hence, slowing down the computation time.

5.5.1 Scaling over atoms

Here, we present the scaling studies for the 4 mentioned systems on a single node of Marconi-100, using 1-128 threads. This is the parallelisation over number of atoms.

For all systems except the water-curcumin one, we present the full range of analysis. For water-curcumin, this shows the analysis over a 10 ns range (21-30 ns), since as mentioned earlier, it is a dynamically evolving system, and therefore, we are interested in mean first-passage times in certain ranges.

Figure 5.1: Performance scaling plot for the bulk water system which has a simulation duration of 2 ns, and a system size of 1019 atoms.

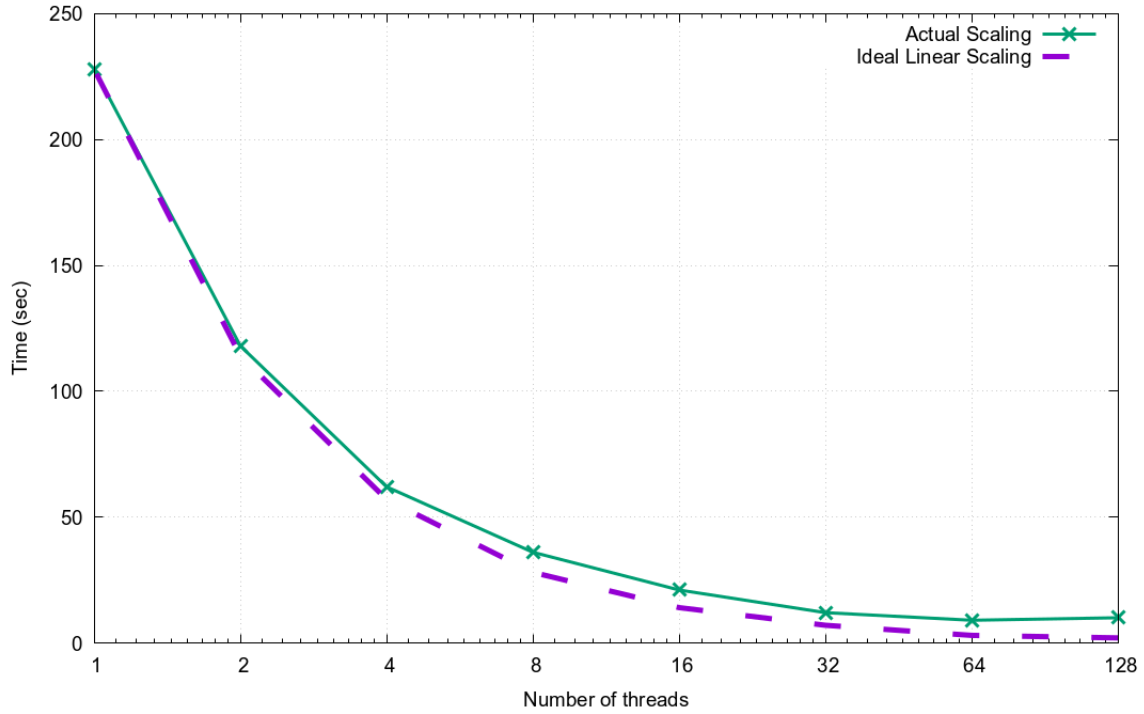


Figure 5.2: Performance scaling plot for the water-air interface system which has a simulation duration of 2 ns, and a system size of 1019 atoms.

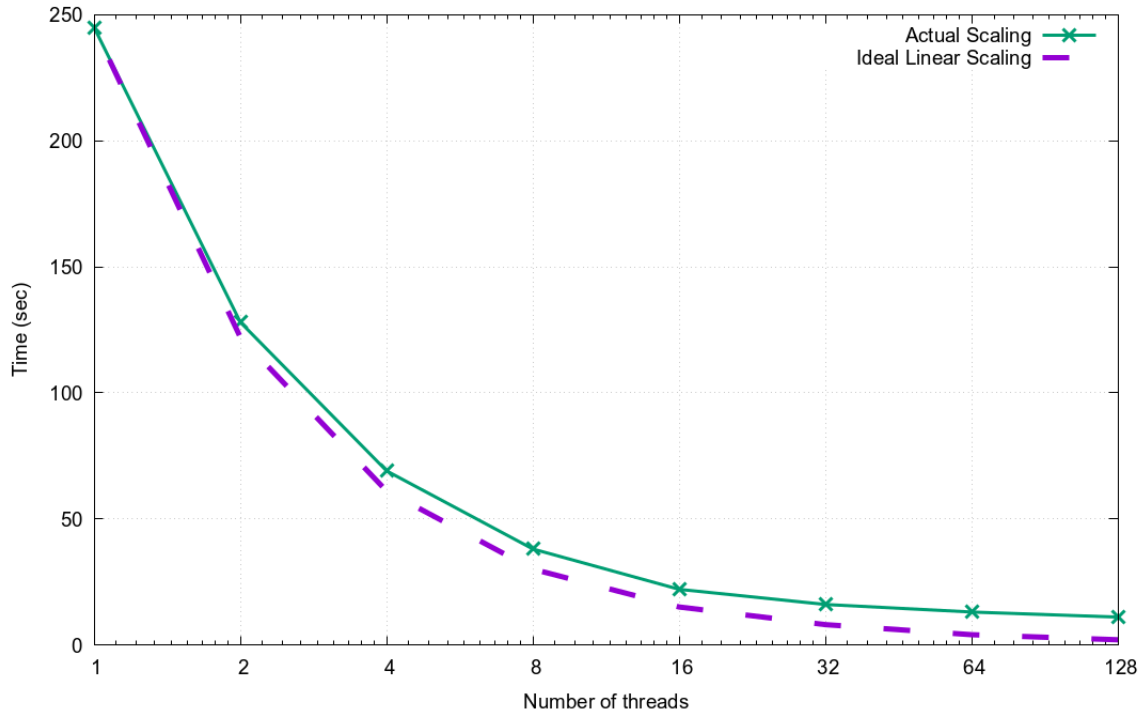


Figure 5.3: Performance scaling plot for the water-C12E6 interface system which has a simulation duration of 2 ns, and a system size of 8086 atoms.

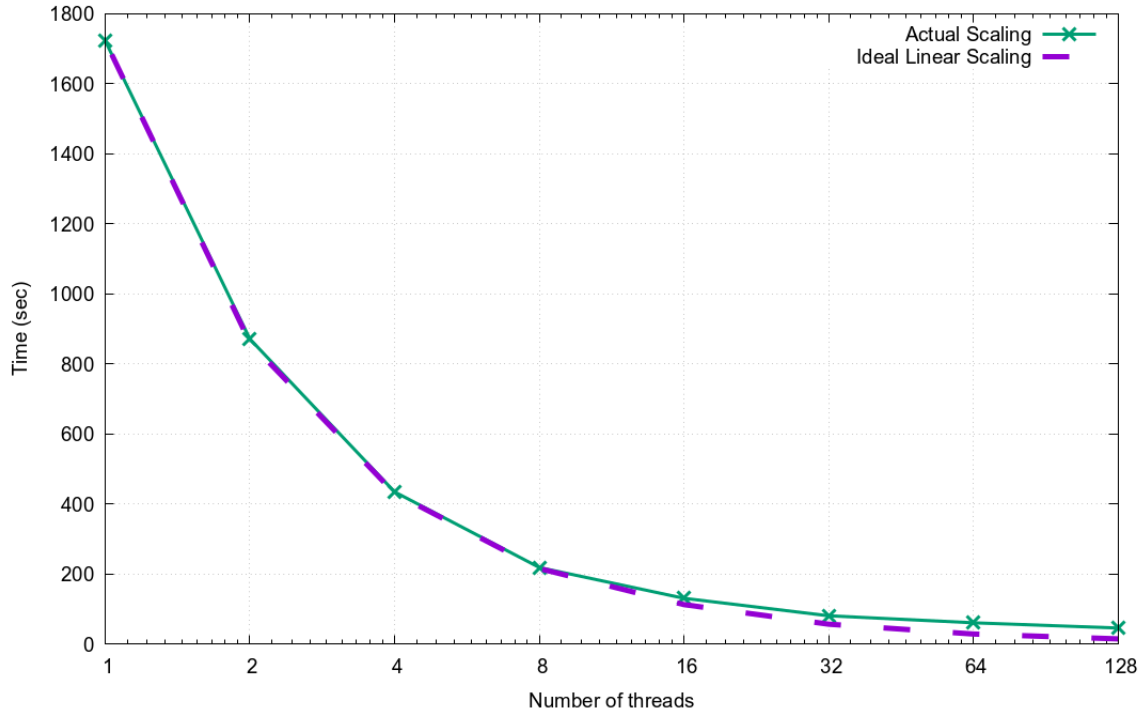
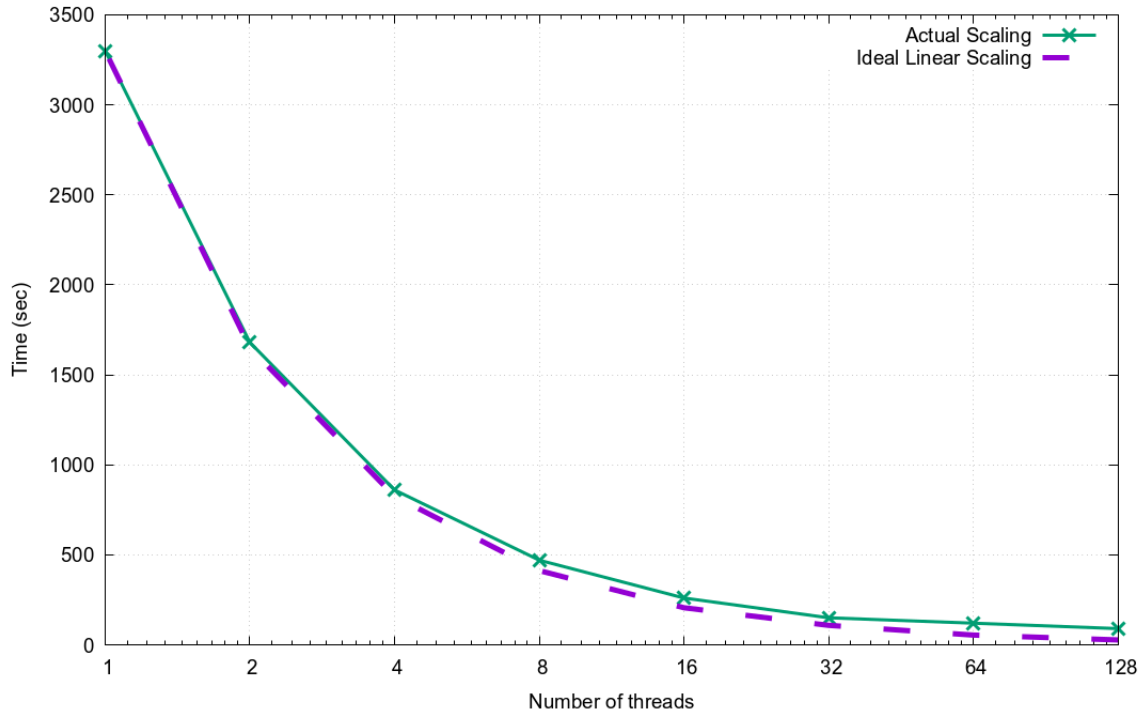


Figure 5.4: Performance scaling plot for the water-curcumin interface system which has a simulation duration of 10 ns, and a system size of 5821 atoms.

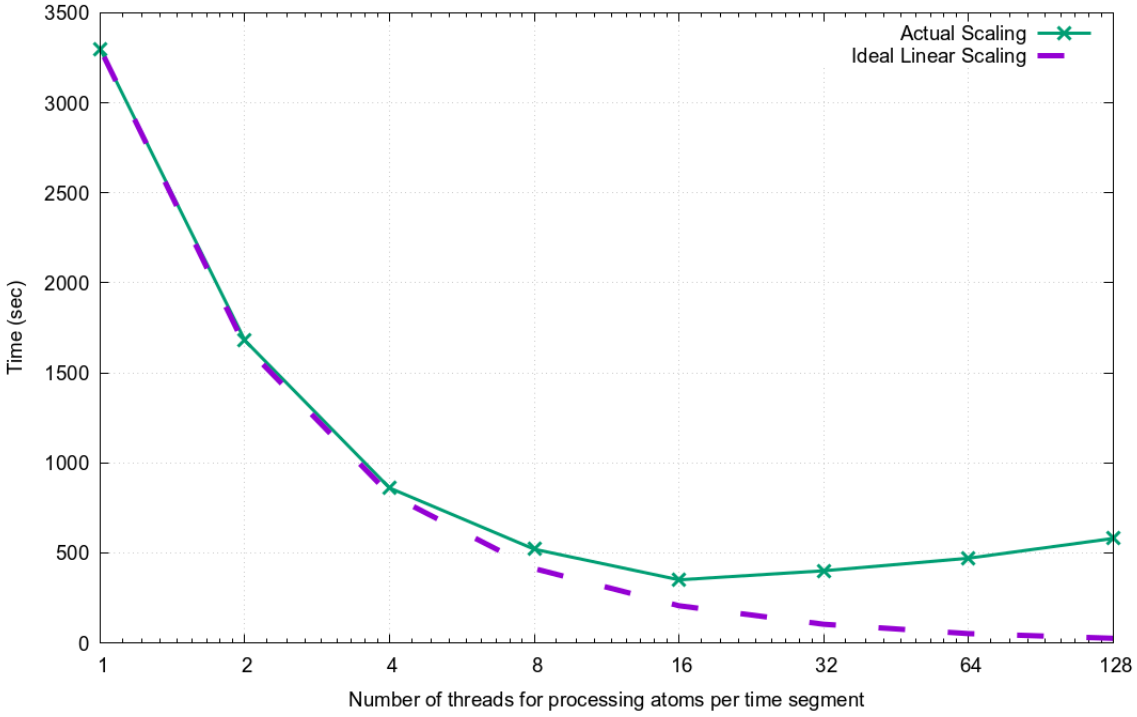


We see that for the larger systems (Water-C12E6 Interface and Water-Curcumin Interface) in Fig.5.3 and Fig.5.4, we get very close to ideal linear scaling up to the 32 physical cores. The extra threads due to hyperthreading still show good scaling, but there is notable deviation from the ideal linear behaviour. This is in line with expectations since the problem of scaling over atoms, apart from load balancing, is embarrassingly parallel.
For the 2 smaller systems (Bulk Water and Water-Air Interface) in Fig.5.1 and Fig.5.2, we see a greater degree of deviation from ideality as we increase the number of threads. This is mostly due to a degree of "diminishing returns" with increased scaling as the time to create all the processes and other typical overhead associated with parallelism becomes significant compared to the very small total execution time. The much larger overall execution time of Water-C12E6 Interface and Water-Curcumin Interface make this a relatively lesser bottleneck in those cases.

5.5.2 Scaling over time

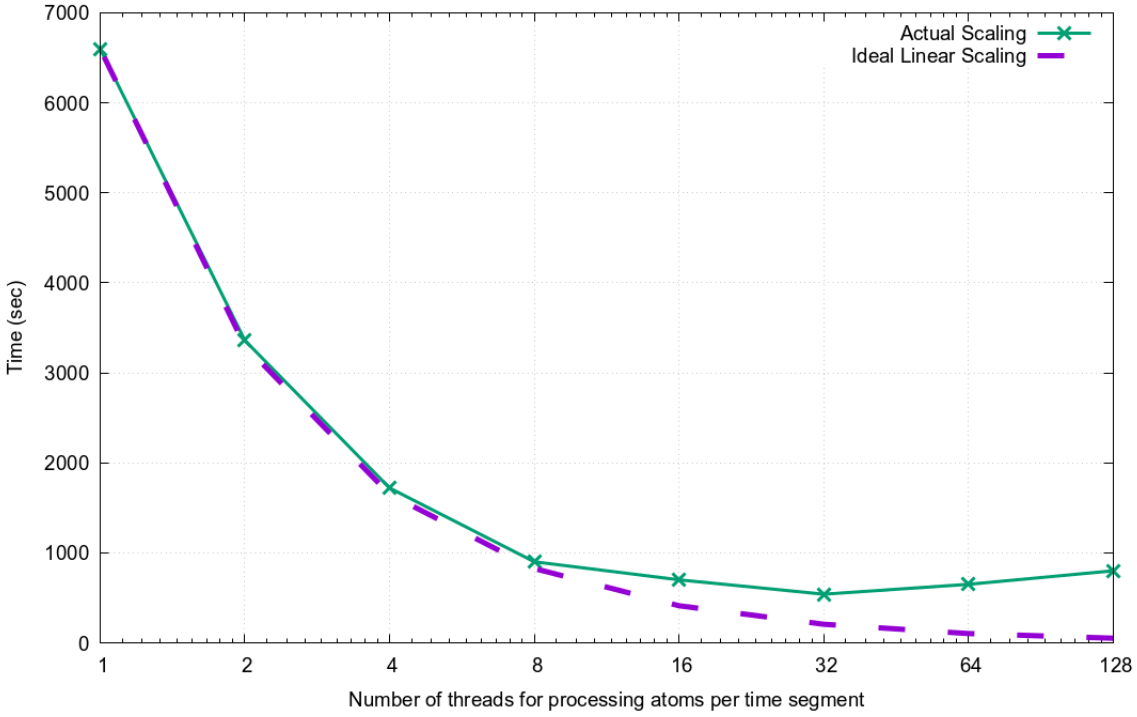
Here, we present the scaling studies for the Water-Curcumin Interface system on a single node of Marconi-100, using up to 128 threads. This is the parallelisation over time, where we also parallelise over atoms in each time-segment, thereby creating a second level of parallelisation.
Since the water-curcumin system is dynamically evolving, we are interested in diffusive dynamics over certain time segments.
Here, we consider the 10 ns segments from 21-100 ns (ignoring the first 20 ns for equilibriation purposes). Each of these 8 segments are parallelised over the total number of atoms as well - using 1-128 threads per segment, thereby resulting in a maximum of 1024 created processes.

Figure 5.5: Performance scaling plot for the water-curcumin interface system which has an overall simulation duration of 100 ns, and a system size of 5821 atoms. The first 20 ns has been discarded for equilibration purposes, and the remainder from 21-100 ns are separated in 8 blocks of 10 ns each.



Here, we consider the 20 ns segments from 21-100 ns (ignoring the first 20 ns for equilibriation purposes). Each of these 4 segments are parallelised over the total number of atoms as well - using 1-128 threads per segment, thereby resulting in a maximum of 512 created processes.

Figure 5.6: Performance scaling plot for the water-curcumin interface system which has an overall simulation duration of 100 ns, and a system size of 5821 atoms. The first 20 ns has been discarded for equilibration purposes, and the remainder from 21-100 ns are separated in 4 blocks of 20 ns each.



The maximum number of hardware threads available is 128.
In Fig.5.5, the X-axis values must be multiplied by 8 to see the total number of threads being used by the system. As one would expect, till 16 threads per time segment, so a total of 128 threads being used by the system, we get decent scaling. The best, almost perfectly linear scaling, is obtained till 4 threads per time segment, so as long as 32 threads are used on the CPU which has 32 physical cores. For 8 and 16 threads per time segment, we still see good (but not linear) scaling. However, when we switch to 32 and higher number of threads per time segment, our performance starts to degrade significantly, as one would expect since we are creating a very large number of processes, and the overhead associated with this parallelism is massive, especially since it doesn't bring any corresponding performance boost due to over-saturating the available hardware.
Similar behaviour is also observed in Fig.5.6.
Since we were interested in behaviour over 10 ns chunks, we chose the optimal combination of 8 time segments, and 16 threads per time segment to process all the atoms, to get the best possible performance.

Chapter 6

Conclusions and Future Work

In this thesis, we developed a toolbox to analyse first-passage times and other associated quantities, and subsequently, the space-dependent diffusion coefficient using the theory presented in Chapter 2.
The major scientific goal was to use the theoretical methods developed in Ref.[2], in combination with molecular dynamics, to study the diffusive dynamics of soft-matter interfaces which are incredibly relevant in understanding the behaviour of important biological systems. Such systems are omnipresent in nature, and much is still not understood about their properties and dynamics. In this thesis, we have used the first-passage analysis method to great effect to gain physical understanding of a variety of well-established systems, reproducing prior work from Refs.[1] and [2]. Finally, we have studied the novel curcumin system, to gain previously unknown insight into its behaviour.
The primary computational goal was to ensure portability, and ease-of-use by non-programmers. To that end, our software can be run in a simple Anaconda environment on any Unix-based system just by editing a plain text file to choose your desired options. This software will also soon be available on GitHub so that it can be used easily by the wider molecular dynamics and statistical physics community.
Our software also ensures the automated production of a wide variety of interesting plots, with/without smoothing the curves as the user may decide.
There is also built-in parallelisation which can work on anything from simple laptops to a node on a computational cluster. The default parallelisation is designed to work over the total number of atoms, and is memory bound. There is an alternative algorithm which is much slower but works independent of system memory.
We show results and benchmarks from a variety of systems, reproducing prior work with well-established ground truths. The software also works on both wrapped (PBC) and unwrapped trajectories. It also allows the user to choose an axis of binning and a separate axis along which to measure the FPT statistics, so that one may get insight into the diffusive dynamics of the system in directions non-orthogonal to the surface.
The user can also choose to parallelise over segments of the available molecular dynamics trajectory, for very large simulations, especially ones with evolving behaviour. We show an example of this in our novel, preliminary results in the water-curcumin system. The feature to parallelise over segments of time allows this methodology to be used as a diagnostic tool to study equilibration.
We will continue working on this software and potential future improvements will include

- Rolling memory loading of the trajectory - Read parts of the trajectory into memory depending on available system memory and desired degree of parallelisation
- first-passage times for cubes such as, Water-Glutamine system from Ref.[1]
- Inferring the potential from the first-passage dynamics using the theory given in Ref.[2]

Through this thesis, we present the theoretical and computational methodology to study heterogeneous soft-matter systems and their diffusive dynamics, in an efficient and user-friendly way.

Bibliography

[1] Roman Belousov, Muhammad Nawaz Qaisrani, Ali Hassanali, and Edgar Roldán. First-passage fingerprints of water diffusion near glutamine surfaces. Soft Matter, 16(40):9202–9216, October 2020. Publisher: The Royal Society of Chemistry.

[2] Roman Belousov, Ali Hassanali, and Édgar Roldán. Statistical physics of inhomogeneous transport: Unification of diffusion laws and inference from first-passage statistics. Phys. Rev. E, 106:014103, Jul 2022.

[3] A. Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. Annalen der Physik, 322(8):549–560, 1905.

[4] Nira Richter-Dyn Narendra S. Goel. Stochastic Models in Biology. Academic Press, 1974.

[5] Sidney Redner. A Guide to First-Passage Processes. Cambridge University Press, 2001.

[6] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. Journal of Computational Chemistry, 32(10):2319–2327, apr 2011.

[7] Richard Gowers, Max Linke, Jonathan Barnoud, Tyler Reddy, Manuel Melo, Sean Seyler, Jan Domański, David Dotson, Sébastien Buchoux, Ian Kenney, and Oliver Beckstein. MDAnalysis: A python package for the rapid analysis of molecular dynamics simulations. In Proceedings of the Python in Science Conference. SciPy, 2016.

[8] Ozge Engin, Alessandra Villa, Mehmet Sayar, and Berk Hess. Driving forces for adsorption of amphiphilic peptides to the airwater interface. The Journal of Physical Chemistry B, 114(34):11093–11101, 2010. PMID: 20687527.

[9] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma. The missing term in effective pair potentials. The Journal of Physical Chemistry, 91(24):6269–6271, 1987.

[10] Giovanni Bussi, Davide Donadio, and Michele Parrinello. Canonical sampling through velocity-rescaling. The Journal of Chemical Physics, 126(1):014101, January 2007.

[11] William L. Jorgensen and Julian Tirado-Rives. Potential energy functions for atomic-level simulations of water and organic and biomolecular systems. Proceedings of the National Academy of Sciences, 102(19):6665–6670, 2005.

[12] Dask Development Team. Dask: Library for dynamic task scheduling, 2016.

[13] William G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. Phys. Rev. A, 31:1695–1697, Mar 1985.

[14] A. W. C. Lau and T. C. Lubensky. State-dependent diffusion: Thermodynamic consistency and its path integral formulation. Phys. Rev. E, 76:011123, Jul 2007.

[15] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. Computing in Science & Engineering, 13(2):22–30, 2011.

[16] Pu Liu, Edward Harder, and B. J. Berne. On the calculation of diffusion coefficients in confined fluids and interfaces with an application to the liquidvapor interface of water. The Journal of Physical Chemistry B, 108(21):6595–6602, 2004.

[17] Pu Liu, Edward Harder, and B. J. Berne. Hydrogen-bond dynamics in the airwater interface. The Journal of Physical Chemistry B, 109(7):2949–2955, 2005. PMID: 16851308.

[18] Gerhard Hummer. Position-dependent diffusion coefficients and free energies from bayesian analysis of equilibrium and replica molecular dynamics simulations. New Journal of Physics, 7(1):34, jan 2005.

[19] Wilmer Olivares-Rivas, Pedro J. Colmenares, and Floralba López. Direct evaluation of the position dependent diffusion coefficient and persistence time from the equilibrium density profile in anisotropic fluids. The Journal of Chemical Physics, 139(7):074103, 2013.